RTI Federation Service

User's Manual

Version 4.5

Your systems. Working as one.

© 2012 Real-Time Innovations, Inc.
All rights reserved.
Printed in U.S.A. First printing.
March 2012.

Trademarks

Real-Time Innovations, RTI, and Connext are trademarks or registered trademarks of Real-Time Innovations, Inc. All other trademarks used in this document are the property of their respective owners.

Copy and Use Restrictions

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form (including electronic, mechanical, photocopy, and facsimile) without the prior written permission of Real-Time Innovations, Inc. The software described in this document is furnished under and subject to the RTI software license agreement. The software may be used or copied only under the terms of the license agreement.

Technical Support

Real-Time Innovations, Inc.
232 E. Java Drive
Sunnyvale, CA 94089
Phone: (408) 990-7444
Email: support@rti.com
Website: https://support.rti.com/

Contents

1 Welcome to RTI Federation Service

1.1 Available Documentation 1-2

2 Configuring Federation Service

2.1 Terms to Know 2-2

2.2 How to Load the XML Configuration 2-2

2.3 XML Syntax and Validation 2-3

2.4 XML Tags for Configuring Federation Service 2-6

2.4.1 Federation Service 2-6

2.4.2 Domain Route 2-9

2.4.3 Administration 2-10

2.4.4 Monitoring 2-12

2.4.5 Session 2-22

2.4.6 Topic Routes 2-24

2.4.7 Auto Topic Routes 2-32

2.5 Enabling and Disabling Federation Service Entities 2-37

3 Running Federation Service

3.1 Starting Federation Service 3-1

3.2 Stopping Federation Service 3-1

4 Administering Federation Service from a Remote Location

4.1 Enabling Remote Administration 4-1

4.2 Remote Commands 4-2

4.2.1 add_peer 4-4

4.2.2 create 4-4

4.2.3 delete 4-5

4.2.4 disable 4-5

4.2.5 enable 4-6

4.2.6 load 4-6

4.2.7 pause 4-6

4.2.8 resume 4-7

4.2.9 save 4-7

4.2.10 unload ... 4-7
4.2.11 update ... 4-7
4.3 Accessing Federation Service from Connext Applications ... 4-10

5 Monitoring Federation Service from a Remote Location

5.1 Enabling Remote Monitoring ... 5-2
5.2 Monitoring Configuration Data ... 5-2
5.2.1 Configuration Data for the Federation Service ... 5-3
5.2.2 Configuration Data for a Domain Route ... 5-4
5.2.3 Configuration Data for a Session ... 5-6
5.2.4 Configuration Data for a Topic Route ... 5-7
5.2.5 Configuration Data for an Auto Topic Route ... 5-11
5.3 Monitoring Status ... 5-15
5.3.1 How the Statistics are Generated ... 5-16
5.3.2 Status Information for the Federation Service ... 5-17
5.3.3 Domain Route Status ... 5-18
5.3.4 Status Information for a Session ... 5-20
5.3.5 Status Information for a Topic Route ... 5-21
5.3.6 Status Information for an Auto Route ... 5-23

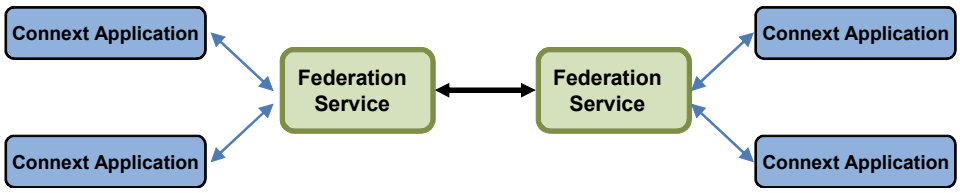
6 Traversing Wide Area Networks

6.1 TCP Communication Scenarios ... 6-2
6.1.1 Communication Within a Single LAN ... 6-2
6.1.2 Symmetric Communication Across NATs ... 6-3
6.1.3 Asymmetric Communication Across NATs ... 6-5
6.1.4 Secure Communication ... 6-6
6.2 Configuring the TCP Transport ... 6-6
6.2.1 TCP Transport Initial Peers ... 6-7
6.2.2 Setting Up the TCP Transport Properties with the PropertyQoSPolicy ... 6-8
6.2.3 TCP/TLS Transport Properties ... 6-9

Chapter 1 Welcome to RTI Federation Service

Welcome to *RTI® Federation Service*. This component of *RTI Connext™ Messaging* is an out- of-the-box solution for federating geographically dispersed systems. It scales *RTI Connext* (formerly, *RTI Data Distribution Service*) applications across domains, LANs and WANs, including firewall and NAT traversal.

Traditionally, *Connext* applications can only communicate with applications in the same domain. With *Federation Service*, you can send and receive data across domains. You can also control which data is sent by using allow and deny lists.



Simply set up *Federation Service* to pass data from one domain to another and specify any desired data filtering. No changes are required in the *Connext* applications.

Key benefits of *Federation Service*:

- **It can significantly reduce the time and effort spent integrating and scaling Connext applications across Wide Area Networks and Systems-of-Systems.**

  Many systems today already rely on *Connext* to distribute their information across a Local Area Network (LAN). However, more and more of these systems are being integrated in Wide Area Networks (WANs). With *Federation Service*, you can scale *Connext* real-time publish/subscribe data-distribution beyond the current local networks and make it available throughout a WAN.

- **With *Federation Service*, you can build modular systems out of existing systems.** Data can be contained in private domains within subsystems and you can designate that only certain "global topics" can be seen across domains. The same mechanism controls the scope of discovery. Both application-level and discovery traffic can be scoped, facilitating scalable designs.
- ***Federation Service* provides secure deployment across multiple sites.** You can partition networks and protect them with firewalls and NATS and precisely control the flow of data between the network segments.
- ***Federation Service* provides features for development, integration and testing.** Multiple sites can each locally test and integrate their core application, expose selected topics of data, and accept data from remote sites to test integration connectivity, topic compatibility and specific use-cases.
- **It connects remotely to live, deployed systems** so you can perform live data analytics, fault condition analysis, and data verification.

1.1 Available Documentation

Federation Service documentation includes:

- **Getting Started Guide** (RTI_Federation_Service_GettingStarted.pdf)—Highlights the benefits of *Federation Service*. It provides installation and startup instructions, and walks you through several examples so you can quickly see the benefits of using *Federation Service*.
- **Release Notes** (RTI_Federation_Service_ReleaseNotes.pdf)—Describes system requirements and compatibility, as well as any version-specific changes and known issues.
- **User's Manual** (RTI_Federation_Service_UsersManual.pdf)—Describes how to configure *Federation Service* and use it remotely.

Chapter 2 Configuring Federation Service

This document describes how to configure *Federation Service*. To see installation and start-up instructions, or to walk through some simple examples, please see the *Getting Started Guide*.

When you start *Federation Service*, you can specify a configuration file in XML format (it is not required). In that file, you can set properties that control the behavior of the service. This chapter describes how to write a configuration file.

This chapter describes:

- ❏ Terms to Know (Section 2.1)
- ❏ How to Load the XML Configuration (Section 2.2)
- ❏ XML Syntax and Validation (Section 2.3)
- ❏ XML Tags for Configuring Federation Service (Section 2.4)
- ❏ Enabling and Disabling Federation Service Entities (Section 2.5)

2.1 Terms to Know

Before learning how to configure *Federation Service*, you should become familiar with a few key terms and concepts.

- A *federation service* entity refers to an execution of *Federation Service*.
- A *domain route* defines a two-way mapping between two specific domains.
- A *session* entity defines a single-threaded context for topic routes.
- A *topic route* defines a one-way mapping between an "input" topic in one domain and an "output" topic in the other domain.
- An *auto topic route* defines a set of potential topic routes based on deny/allow filters on the topic name and registered type name.

2.2 How to Load the XML Configuration

Federation Service loads its XML configuration from multiple locations. This section presents the various approaches, listed in load order.

The first three locations only contain QoS Profiles and are inherited from *Connext* (see Chapter 15 in the *RTI Core Libraries and Utilities User's Manual*).

- **$NDDSHOME/resource/qos_profiles_4.5x[1]/xml/NDDS_QOS_PROFILES.xml**

  This file contains the *Connext* default QoS values; it is loaded automatically if it exists. *(First to be loaded.)*

- File in NDDS_QOS_PROFILES

  The files (or XML strings) separated by semicolons referenced in this environment variable are loaded automatically.

- **<*working directory*>/USER_QOS_PROFILES.xml**

  This file is loaded automatically if it exists.

1. *x* stands for the version letter of the current release.

The next locations are specific to *Federation Service*:

- ***<Federation Service executable location>/../../resource/xml/*** **RTI_FEDERATION_SERVICE.xml**

  This file contains the default *Federation Service* configuration; it is loaded if it exists. **RTI_FEDERATION_SERVICE.xml** defines a service that automatically routes all types and topics between domains 0 and 1.

- ***<working directory>*****/USER_FEDERATION_SERVICE.xml**

  This file is loaded automatically if it exists.

- File specified using the command line parameter -cfgFile

  The command-line option **-cfgFile** (see Table 3.1 in the *Getting Started Guide*) can be used to specify a configuration file.

- File specified using the remote command 'load'

  The **load** command (see Section 4.2.6) allows loading an XML file remotely. The file loaded using this command replaces to the file loaded using the **-cfgFile** command-line option. *(Last to be loaded.)*

You may use a combination of the above approaches.

Figure 2.1 shows an example configuration file. You will learn the meaning of each line as you read the rest of this chapter.

2.3 XML Syntax and Validation

The XML configuration file must follow these syntax rules:

- The syntax is XML; the character encoding is UTF-8.
- Opening tags are enclosed in **<>**; closing tags are enclosed in **</>**.
- A tag value is a UTF-8 encoded string. Legal values are alphanumeric characters. The federation service's parser will remove all leading and trailing spaces[1] from the string before it is processed.

  For example, " **<tag> value </tag>**" is the same as "**<tag>value</tag>**".
- All values are case-sensitive unless otherwise stated.
- Comments are enclosed as follows: **<!-- comment -->**.
- The root tag of the configuration file must be **<dds>** and end with **</dds>**.

Figure 2.1 **Example XML Configuration File**

```
<?xml version="1.0"?>
<dds>
   <federation_service name="BridgeExample" group_name="MyGroup">
        <domain_route name="DomainRoute">
            <participant_1>
               <domain_id>0</domain_id>
            </participant_1>

            <participant_2>
                <domain_id>1</domain_id>
            </participant_2>

            <session name="Session">
                <topic_route name="SquaresBridge">

                    <input participant="1">
                        <registered_type_name>ShapeType</registered_type_name>
                        <topic_name>Square</topic_name>
                    </input>

                    <output>
                        <registered_type_name>ShapeType</registered_type_name>
                        <topic_name>Square</topic_name>
                    </output>

                </topic_route>
            </session>
        </domain_route>
   </federation_service>
</dds>
```

This file configures a simple bridge from domain 0 to domain 1 for Squares.. You will find this example in <Federation Service installation directory>/example/shapes/topic_bridge.xml. Additional examples are in the same directory.

Federation Service provides DTD and XSD files that describe the format of the XML content. We recommend including a reference to one of these documents in the XML file that contains the routine service's configuration—this provides helpful features in code

1. Leading and trailing spaces in enumeration fields will not be considered valid if you use the distributed XSD document to do validation at run-time with a code editor.

editors such as Visual Studio and Eclipse, including validation and auto-completion while you are editing the XML file.

The DTD and XSD definitions of the XML elements are in <*Federation Service installation directory*>/resource/schema/rti_federation_service.dtd and <*Federation Service installation directory*>/resource/schema/rti_federation_service.xsd, respectively.

To include a reference to the XSD document in your XML file, use the attribute **xsi:noNamespaceSchemaLocation** in the **<dds>** tag. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation= "<installation directory for Federation
Service>/resource/schema/rti_federation_service.xsd">
   ...
</dds>
```

To include a reference to the DTD document in your XML file, use the **<!DOCTYPE>** tag.

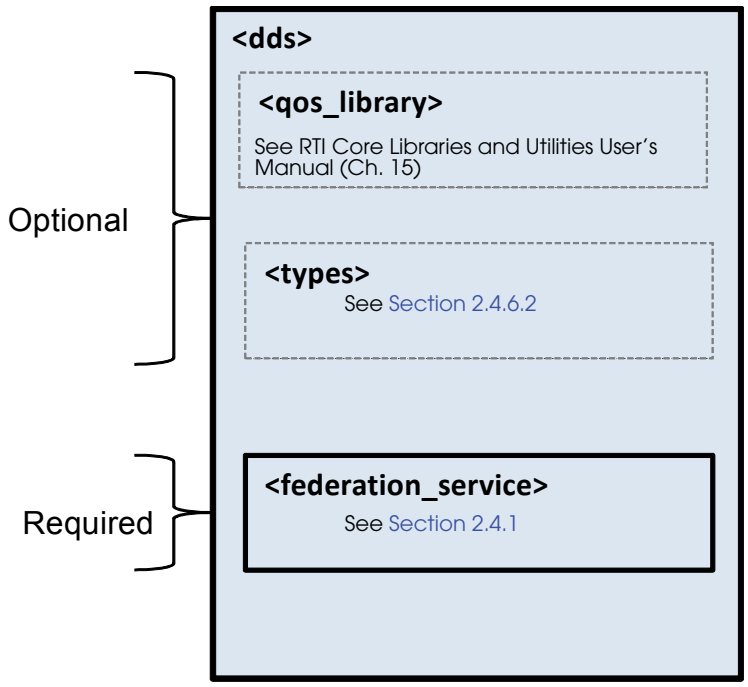
For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE dds SYSTEM "<installation directory for Federation Service>
/resource/schema/rti_federation_service.dtd">
<dds>
   ...
</dds>
```

We recommend including a reference to the XSD file in the XML documents; this provides stricter validation and better auto-completion than the corresponding DTD file.

2.4 XML Tags for Configuring Federation Service

This section describes the XML tags you can use in a *Federation Service* configuration file. The following diagram and Table 2.1 describe the top-level tags allowed within the root **<dds>** tag.



2.4.1 Federation Service

A configuration file must have at least one **<federation_service>** tag; this tag is used to configure an execution of *Federation Service*. A configuration file may contain multiple **<federation_service>** tags.

When you start *Federation Service*, you can specify which **<federation_service>** tag to use to configure the service using the **-cfgName** command-line parameter.

Table 2.1 **Top-level Tags in the Configuration File**

Tags within <dds>	Description	Number of Tags Allowed
<federation_service>	Specifies a *Federation Service* configuration. See Federation Service (Section 2.4.1).	1 or more ***(required)***
<qos_library>	Specifies a QoS library and profiles. The contents of this tag are specified in the same manner as for a *Connext* QoS profile file—see Chapter 15 in the *RTI Core Libraries and Utilities User's Manual*.	0 or more
<types>	Defines the types to be used by the federation service. See Defining Types in the Configuration File (Section 2.4.6.2).	0 or 1

For example:

```
<dds>
    <federation_service name="Bridge1" group_name="Group1">
    ...
    </federation_service>

    <federation_service name="Bridge2" group_name="Group1">
    ...
    </federation_service>
</dds>
```

Starting *Federation Service* with the following command will use the **<federation_service>** tag with the name **Bridge1**:

```
rtifederationservice -cfgFile example.xml -cfgName Bridge1
```

Because a configuration file may contain multiple **<federation_service>** tags, one file can be used to configure multiple *Federation Service* executions.

A federation service belongs to a group of one or more federation services identified by a common **group_name**. Federation services in the same group will *not* communicate with each other.

If the **<federation_service>** tag does not have a **group_name** attribute, *Federation Service* will use the name: **RTI_FederationService_<*Host Name*>_<*Process ID*>**, such as **RTI_FederationService_myhost_20024**.

Table 2.2 describes the tags allowed within a **<federation_service>** tag.

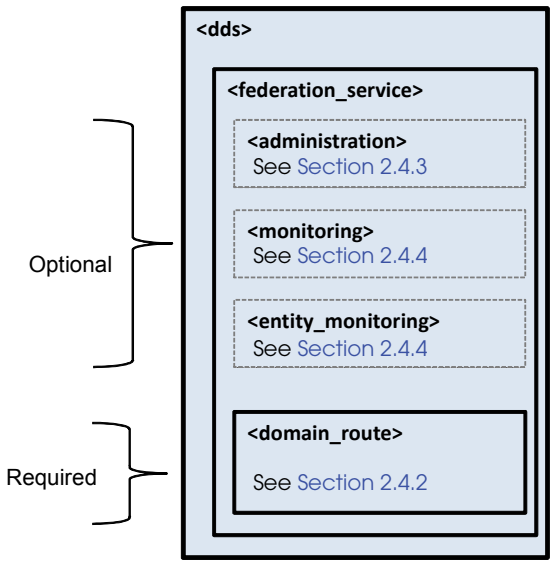


Table 2.2 **Federation Service Tags**

Tags within <federation_service>	Description	Number of Tags Allowed
<annotation>	Contains a <documentation> tag that can be used to provide a federation service description. This description will show up when you run *Federation Service* without the **-cfgName** command-line option.	0 or 1
<domain_route>	Defines a mapping between two domains. See Section 2.4.2.	1 or more ***(required)***
<administration>	Enables and configures remote administration. See Administration (Section 2.4.3) and Chapter 4: Administering Federation Service from a Remote Location.	0 or 1
<monitoring>	Enables and configures general remote monitoring. General monitoring settings are applicable to all the *Federation Service* entities unless they are explicitly overridden. See Monitoring (Section 2.4.4) and Chapter 5: Monitoring Federation Service from a Remote Location.	0 or 1
<entity_monitoring>	Enables and configures remote monitoring for the federation_service entity.	0 or 1

2.4.2 Domain Route

A *domain route* defines a mapping between two specific domains. Data published in either of these domains can be routed to the other one.

A domain route creates two participants, known as **participant_1** and **participant_2**. Each participant belongs to one of two specified domains.

For example:

```
<dds>
    <federation_service name="Router1"
      group_name="Group1">
        <domain_route name="DomainRoute1">
            <participant_1>
                <domain_id>54</domain_id>
                ...
            </participant_1>

            <participant_2>
                <domain_id>55</domain_id>
                ...
            </participant_2>

            <session name="Session">
            ...
            </session>
        </domain_route>
        ...
    </federation_service>
</dds>
```

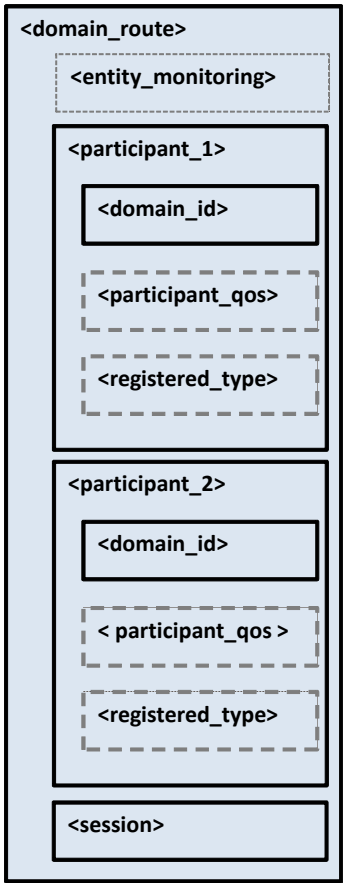


Table 2.3 lists the tags allowed within a **<domain_route>** tag.

Table 2.3 **Domain Route Tags**

Tags within <domain_route>	Description	Number of Tags Allowed
<participant_1>	Configures the first participant. See Table 2.4.	1 ***(required)***
<participant_2>	Configures the second participant. See Table 2.4.	1 ***(required)***

Table 2.3 **Domain Route Tags**

Tags within <domain_route>	Description	Number of Tags Allowed
<session>	Defines a single-threaded context in which data is routed according to specified topic routes. See Session (Section 2.4.5).	1 or more ***(required)***
<entity_monitoring>	Enables and configures remote monitoring for the domain route. See Monitoring (Section 2.4.4) and Chapter 5: Monitoring Federation Service from a Remote Location.	0 or 1

Table 2.4 lists the tags allowed within **<participant_1>** and **<participant_2>** tags.

2.4.3 Administration

You can create a *Connext* application that can remotely control *Federation Service*. The **<administration>** tag is used to enable remote administration and configure its behavior.

By default, remote administration is turned off in *Federation Service* for security reasons. A remote administration section is not required in the configuration file.

For example:

```
<dds>
    <federation_service>
        <administration>
            <domain_id>55</domain_id>
           <save_path>/home/david/mysaved_config.xml</save_path>
        </administration>
        ...
    </federation_service>
</dds>
```

When remote administration is enabled, *Federation Service* will create a DomainParticipant, Publisher, Subscriber, DataWriter, and DataReader. These entities are used to receive commands and send responses. You can configure these entities with QoS tags within the **<administration>** tag.

Table 2.5 lists the tags allowed within **<administration>** tag.

For more details, please see Chapter 4: Administering Federation Service from a Remote Location.

Note: The command-line options used to configure remote administration take precedence over the XML configuration (see Table 3.1 in the *Getting Started Guide*).

Table 2.4 **Participant Tags**

Tags within <participant_1/2>	Description	Number of Tags Allowed
<domain_id>	Sets the domain ID associated with the participant.	1 ***(required)***

<memory_management>	Configures certain aspects of how *Connext* allocates internal memory. The configuration is per domain_route's participant and therefore affects all the contained DataReaders and DataWriters. For example:
	`<domain_route name="test">`
	`  <participant_1>`
	`  <domain_id>0</domain_id>`
	`  ...`
	`  **<memory_management>**`
	`  <sample_buffer_min_size>`
	`  X`
	`  </sample_buffer_min_size>`
	`  <sample_buffer_trim_to_size>`
	`  true`
	`  </sample_buffer_trim_to_size>`
	`  </memory_management>`
	`  </participant_1>`
	`  ...`
	The <memory_management> tag can include the following tags:
	❑ **sample_buffer_min_size**: For all DataReaders/DataWriters, the way Connext allocates memory for samples is as follows: Connext pre-allocates space for samples up to size X in the reader and writer queues. If a sample has an actual size greater than X, the memory is allocated dynamically for that sample. The default size is DDS_LENGTH_UNLIMITED (meaning no dynamic memory is used; the maximum sample size is pre-allocated).
	❑ **sample_buffer_trim_to_size**: If set to true, after allocating dynamic memory for very large samples, that memory will be released when possible. If false, that memory will not be released but kept for future samples if needed. The default is false.
	This feature is useful when a data type has a very high maximum size (e.g., megabytes) but most of the samples sent are much smaller than the maximum possible size (e.g., kilobytes). In this case, the memory footprint is reduced dramatically, while still correctly handling the rare cases in which very large samples are published.
		0 or 1

Table 2.4 **Participant Tags**

Tags within <participant_1/2>	Description	Number of Tags Allowed
<registered_type>	Registers a type name and associates it with a type code. When you define a type in the configuration file (with the **<types>** tag), you have to register the type in order to use it in topic routes. See Topic Route Types (Section 2.4.6.1).	0 or more

<participant_qos>	Sets the participant QoS.
	The contents of this tag are specified in the same manner as a *Connext* QoS profile file—see Chapter 15 in the *RTI Core Libraries and Utilities User's Manual*.
	If not specified, the default is used.
	You can use a <participant_qos> tag inside a library/profile previously defined in your configuration file by referring to it like this:
	`<participant_qos base_name="MyLibrary::MyProfile" />`
	To use that profile but override just some values:
	`<participant_qos base_name="MyLibrary::MyProfile">`
	`  <resource_limits>`
	`  <contentfilter_property_max_length>`
	`  64`
	`  </contentfilter_property_max_length>`
	`  </resource_limits>`
	`</participant_qos>`
	(This applies to all QoS tags: **<publisher_qos>**, **<subscriber_qos>** in sessions; **<datareader_qos>**, **<datawriter_qos>** in topic routes and auto topic routes.)
		0 or 1

2.4.4 Monitoring

You can create a *Connext* application that can remotely monitor the status of *Federation Service*. To enable remote monitoring and configure its behavior, use the **<monitoring>** and **<entity_monitoring>** tags.

By default, remote monitoring is turned off in *Federation Service* for security and performance reasons. A remote monitoring section is not required in the configuration file. For more details, please see Chapter 5: Monitoring Federation Service from a Remote Location.

Table 2.5 **Remote Administration Tags**

Tags within <administration>	Description	Number of Tags Allowed
<domain_id>	Specifies the domain ID that *Federation Service* will use to enable remote administration.	1 (*Required*)
<participant_qos>	Configures the DomainParticipant QoS for remote administration. If the tag is not defined, *Federation Service* will use the *Connext* defaults.	0 or 1
<publisher_qos>	Configures the Publisher QoS for remote administration. If the tag is not defined, *Federation Service* will use the *Connext* defaults.	0 or 1
<subscriber_qos>	Configures the Subscriber QoS for remote administration. If the tag is not defined, *Federation Service* will use the *Connext* defaults.	0 or 1
<datareader_qos>	Configures the DataReader QoS for remote administration. If the tag is not defined, *Federation Service* will use the *Connext* defaults with the following changes: reliability.kind = DDS_RELIABLE_RELIABILITY_QOS (this value cannot be changed) history.kind = DDS_KEEP_ALL_HISTORY_QOS resource_limits.max_samples = 32	0 or 1
<datawriter_qos>	Configures the DataWriter QoS for remote administration. If the tag is not defined, *Federation Service* will use the *Connext* defaults with the following changes: history.kind = DDS_KEEP_ALL_HISTORY_QOS resource_limits.max_samples = 32	0 or 1
<save_path>	Specifies the file that will contain the saved configuration. It is empty by default. A <save_path> must be specified if you want to use the save (Section 5.2.9) command. If the file specified by <save_path> already exists, the file will be overwritten when **save** is executed. This value is mutable when an update (Section 5.2.11) command targets a federation service. This value is sent as part of the monitoring configuration data for the federation service (see Configuration Data for the Federation Service (Section 5.2.1)).	0 or 1

For example:

```
<dds>
    <federation_service>
        <enabled>true</enabled>
        <monitoring>
            <domain_id>55</domain_id>
            <status_publication_period>
                <sec>1</sec>
            </status_publication_period>
        </monitoring>
        ...
    </federation_service>
</dds>
```

Federation Service allows monitoring of the following kinds of entities:

- **<federation_service>** (see Section 2.4.1)
- **<domain_route>** (see Section 2.4.2)
- **<session>** (see Section 2.4.5)
- **<topic_route>** (see Section 2.4.6)
- **<auto_topic_route>** (see Section 2.4.7)

For each entity, *Federation Service* can publish two kinds of information:

- ❏ Entity data
- ❏ Entity status

Entity data provides information about the configuration of the entity. For example, the topic route data contains information such as the topic name and the type name. Entity data information is republished every time the entity is enabled, disabled or has configuration changes.

Entity status provides information about the operational status of an entity. This kind of information changes continuously and is computed and published periodically. For example, the topic route status contains information such as the topic route's latency and throughput.

For more information about entity data and status, see Chapter 5: Monitoring Federation Service from a Remote Location.

When remote monitoring is enabled, *Federation Service* will create one DomainParticipant, one Publisher, five DataWriters for data publication (one for each kind of entity), and five DataWriters for status publication (one for each kind of entity). You can config-

ure the QoS of these entities with the **<monitoring>** tag defined under **<federation_service>**.

The general remote monitoring parameters specified using the **<monitoring>** tag in **<federation_service>** (except **domain_id**, **participant_qos**, **publisher_qos**, and **datawriter_qos**) can be overwritten on a per entity basis using the **<entity_monitoring>** tag.

For example:

```
<dds>
    <federation_service>
        <monitoring>
            <domain_id>55</domain_id>
            <status_publication_period>
                <sec>1</sec>
            </status_publication_period>
        </monitoring>
        ...
        <domain_route>
            <entity_monitoring>
                <status_publication_period>
                    <sec>4</sec>
                </status_publication_period>
            </entity_monitoring>
            ...
        </domain_route>
    </federation_service>
</dds>
```

Table 2.6 lists the tags allowed within **<monitoring>** tag.

2.4.4.1 Monitoring Configuration Inheritance

The monitoring configuration defined in **<federation_service>** is inherited by all the entities defined inside the tag.

An entity can overwrite three elements of the monitoring configuration:

- The status publication period
- The statistics sampling period
- The historical statistics windows

Each one of this three elements is inherited and can be overwritten independently using the **<entity_monitoring>** tag.

For example:

Table 2.6 **Monitoring tags**

Tags within <monitoring>	Description	Number of Tags Allowed
<enabled>	Enables/disables general remote monitoring. Setting this value to true (default value) in the **<monitoring>** tag under **<federation_service>** enables monitoring in all the entities unless they explicitly disable it by setting this tag to false in their local **<entity_monitoring>** tags. Setting this tag to false in the **<monitoring>** tag under **<federation_service>** disables monitoring in all the *Federation Service* entities. In this case, any monitoring configuration settings in the entities are ignored.	0 or 1
<domain_id>	Specifies which domain ID *Federation Service* will use to enable remote monitoring.	1 (*required*)
<participant_qos>	Configures the DomainParticipant QoS for remote monitoring. If the tag is not defined, *Federation Service* will use the *Connext* defaults with the following change: resource_limits.type_code_max_serialized_length = 4096	0 or 1
<publisher_qos>	Configures the Publisher QoS for remote monitoring. If the tag is not defined, *Federation Service* will use the *Connext* defaults.	0 or 1
<datawriter_qos>	Configures the DataWriter QoS for remote monitoring. If the tag is not defined, *Federation Service* will use the *Connext* defaults with the following change: durability.kind = DDS_TRANSIENT_LOCAL_DURABILITY_QOS	0 or 1

Table 2.6 **Monitoring tags**

Tags within <monitoring>	Description	Number of Tags Allowed

<status_publication_period>	Specifies the frequency at which the status of an entity is published.
	For example:
	`<status_publication_period>`
	`<sec>3</sec>`
	`<nanosec>0</nanosec>`
	`</status_publication_period>`
	If the tag is not defined, the period is 5 seconds.
	The status publication period defined in **<federation_service>** is inherited by all the entities inside **<federation_service>**.
	An entity can overwrite the period.
		0 or 1

<statistics_sampling_period>	Specifies the frequency at which status statistics are gathered. Statistical variables such as latency, are part of the entity status. For example:
	`<statistics_sampling_period>`
	`<sec>1</sec>`
	`<nanosec>0</nanosec>`
	`</statistics_sampling_period>`
	The statistics period for a given entity should be smaller than the publication period.
	If the tag is not defined, the period is 1 second.
	The statistics sampling period defined in **<federation_service>** is inherited by all the entities inside **<federation_service>**.
	An entity can overwrite the period.
		0 or 1

Table 2.6 **Monitoring tags**

Tags within <monitoring>	Description	Number of Tags Allowed

<historical_statistics>	Enables or disables the publication of statistics calculated within fixed time windows.
	By default, *Federation Service* only publishes the statistics corresponding to the window between two status publications.
	By using this tag, you can get the following additional windows:
	❑ 5 seconds
	❑ 1 minute
	❑ 5 minutes
	❑ 1 hour
	❑ Up time (since the entity was enabled)
	For example:
	`<historical_statistics>`
	`<five_second>true</five_second>`
	`<one_minute>true</one_minute>`
	`<five_minute>false</five_minute>`
	`<one_hour>true</one_hour>`
	`<up_time>false</up_time>`
	`</historical_statistics>`
	If a window is not present (inside the tag **<historical_statistics>**), it is considered disabled.
	Historical statistics can be overwritten on a per entity basis.
		0 or 1

```
<dds>
    <federation_service name="MonitoringExample">
        <monitoring>
            <domain_id>55</domain_id>
            <status_publication_period>
                <sec>1</sec>
            </status_publication_period>
            <statistics_sampling_period>
                <sec>1</sec>
                <nanosec>0</nanosec>
            </statistics_sampling_period>
        </monitoring>
```

```
            ...
            <domain_route>
                <entity_monitoring>
                    <status_publication_period>
                        <sec>4</sec>
                    </status_publication_period>
                </entity_monitoring>
                ...
            </domain_route>
        </federation_service>
    </dds>
```

In the previous example, the domain route overwrites the status publication period to 4 seconds and inherits the statistics sampling period.

Table 2.7 **Entity Monitoring Tags**

Tags within <entity_monitoring>	Description	Number of Tags Allowed
<enabled>	Enables/disables remote monitoring for a given entity. If general monitoring is disabled this value is ignored. Default value: true	0 or 1
<status_publication_period>	Specifies the frequency at which the status of an entity is published. For example: <status_publication_period> <sec>3</sec> <nanosec>0</nanosec> </status_publication_period> If the tag is not defined, its value is inherited from the general monitoring settings.	0 or 1

Table 2.7 **Entity Monitoring Tags**

Tags within <entity_monitoring>	Description	Number of Tags Allowed

<statistics_sampling_period>	Specifies the frequency at which status statistics are gathered. Statistical variables such as latency, are part of the entity status. For example:
	`<statistics_sampling_period>`
	`  <sec>1</sec>`
	`  <nanosec>0</nanosec>`
	`</statistics_sampling_period>`
	The statistics period for a given entity should be smaller than the publication period.
	If the tag is not defined, the period is inherited from the general monitoring settings.
	This tag is only present in the <entity_monitoring> tag of <topic_route>, <auto_topic_route> and <federation_service>.
		0 or 1

Table 2.7 **Entity Monitoring Tags**

Tags within <entity_monitoring>	Description	Number of Tags Allowed

<historical_statistics>	Enables or disables the publication of statistics calculated within fixed time windows.
	By default, *Federation Service* only publishes the statistics corresponding to the window between two status publications.
	By using this tag, you can get the following additional windows:
	❑ 5 seconds
	❑ 1 minute
	❑ 5 minutes
	❑ 1 hour
	❑ Up time (since the entity was enabled)
	For example:
	`<historical_statistics>`
	`<five_second>true</five_second>`
	`<one_minute>true</one_minute>`
	`<five_minute>false</five_minute>`
	`<one_hour>true</one_hour>`
	`<up_time>false</up_time>`
	`</historical_statistics>`
	If a window is not present (inside the tag **<historical_statistics>**), it is considered disabled.
	If this tag is not defined, historical statistics are inherited from the general monitoring settings.
		0 or 1

2.4.5 Session

A **<session>** tag defines a single-threaded context for data bridging; it may also contain topic routes (Section 2.4.6) and auto-topic routes (Section 2.4.7).

For example:

```
<dds>
    ...
    <federation_service name="MyFederationService">
        ...
        <domain_route>
        ...
            <session name="Session1">
                ...
                <topic_route name="TopicRoute1" >
                    ...
                </topic_route>
                ...
            </session>
            ...
        </domain_route>
        ...
    </federation_service>
    ...
</dds>
```

Each session creates a Publisher and Subscriber on each of the two participants (**participant_1** and **participant_2**).

Table 2.8 lists the tags allowed within a **<session>** tag.

Table 2.8 **Session Tags**

Tags within <session>	Description	Number of Tags Allowed
<auto_topic_route>	Defines a general topic route based on type and topic filters. See Auto Topic Routes (Section 2.4.7).	0 or more
<topic_route>	Defines a data mapping between two topics. See Topic Routes (Section 2.4.6).	0 or more

Table 2.8 **Session Tags**

Tags within <session>	Description	Number of Tags Allowed
<publisher_qos>	Sets the QoS of both Publishers. The contents of this tag are specified in the same manner as a *Connext* QoS profile file—see Chapter 15 in the *RTI Core Libraries and Utilities User's Manual*. If the tag is not defined, *Federation Service* will use the *Connext* defaults.	0 or 1
<subscriber_qos>	Sets the QoS of both Subscribers. The contents of this tag are specified in the same manner as a *Connext* QoS profile file—see Chapter 15 in the *RTI Core Libraries and Utilities User's Manual*. If the tag is not defined, *Federation Service* will use the *Connext* defaults.	0 or 1

<thread>	Sets the mask, priority and stack size of the thread associated with this session.
	Example:
	`<session>`
	`  <thread>`
	`  <mask>MASK_DEFAULT</mask>`
	`  <priority>`
	`  THREAD_PRIORITY_DEFAULT`
	`  </priority>`
	`  <stack_size>`
	`  THREAD_STACK_SIZE_DEFAULT`
	`  </stack_size>`
	`  </thread>`
	`  ...`
	`</session>`
	Default values:
	mask = MASK_DEFAULT
	priority = THREAD_PRIORITY_DEFAULT
	stack_size = THREAD_STACK_SIZE_DEFAULT
		0 or 1

Table 2.8 **Session Tags**

Tags within <session>	Description	Number of Tags Allowed
<wait_set>	Configures the WaitSet used to sleep and notify the thread when data is available. Example: <session> <wait_set> <max_event_count>5</max_event_count> <max_event_delay> <sec>DURATION_INFINITE_SEC</sec> <nanosec>DURATION_INFINITE_NSEC </nanosec> </max_event_delay> </wait_set> ... </session> Default values: max_event_count = 1 max_event_delay.sec = DURATION_INFINITE_SEC max_event_delay.nanosec = DURATION_INFINITE_NSEC	0 or 1
<monitoring>	Enables and configures remote monitoring for the session. See Monitoring (Section 2.4.4) and Chapter 5: Monitoring Federation Service from a Remote Location.	0 or 1

2.4.6 Topic Routes

A topic route explicitly defines a mapping between an "input" topic on one domain and an "output" topic on the other domain.

For example, the following topic route will subscribe to topic Square on domain 54 and will republish those samples on domain 55.

```
<dds>
    ...
    <federation_service>
        ...
        <domain_route>
            <participant_1>
                <domain_id>54</domain_id>
            </participant_1>
```

```
            <participant_2>
                <domain_id>55</domain_id>
            </participant_2>
            ...
            <session name="Session1">
            ...
                <topic_route name="SquaresRoute">
                    <input participant="1">
                        <topic_name>Square</topic_name>
                        <registered_type_name>
                            ShapeType
                        </registered_type_name>
                        ...
                    </input>
                    <output>
                        <topic_name>Square</topic_name>
                        <registered_type_name>
                            ShapeType
                        </registered_type_name>
                        ...
                    </output>
                    ...
                </topic_route>
            </session>
            ...
        </domain_route>
        ...
    </federation_service>
    ...
</dds>
```

The direction is defined by the attribute **participant** of the tag **<input>**. Therefore, to change the above example to read Squares from domain 55 and write them on domain 54, we would have used **<input participant="2">**.

Every topic route creates a DataReader and DataWriter that belong to the corresponding session's Subscriber and Publisher, respectively. The read and write operations will be performed in the context of the thread associated with the session.

Topic Routes vs. Auto Topic Routes: A *topic route* is an explicit route of data for two specific topics. An *auto topic route* (defined with a different tag, **<auto_topic_route>**) is a way to automatically create topic routes based on filters—see Auto Topic Routes (Section 2.4.7).

Table 2.9 lists the tags allowed within a **<topic_route>**.

Table 2.10 lists the tags allowed within the **<input>** and **<output>** tags.

Table 2.9 **Topic Route Tags**

Tags within <topic_route>	Description	Number of Tags Allowed
<input>	Defines the topic route's input topic. See Table 2.10.	1 ***(required)***
<output>	Defines the topic route's output topic. See Table 2.10.	1 ***(required)***
<route_types>	Defines if the input domain will use types discovered in the output domain and vice versa for the creation of DataWriters and DataReaders in the topic route. See Discovering Types (Section 2.4.6.3). Default: false	0 or 1
<publish_with_original_info>	Writes the data sample as if they came from its original writer. Setting this option to true allows having redundant federation services and prevents the applications from receiving duplicate samples. Default: false	0 or 1
<publish_with_ original_timestamp>	When this tag is set to true, the data samples are written with their original source timestamp. Default: false	0 or 1
<propagate_dispose>	Indicates whether or not disposed samples (NOT_ALIVE_DISPOSE) must be propagated by the topic route. Default: true	0 or 1
<propagate_unregister>	Indicates whether or not NOT_ALIVE_NO_WRITERS samples must be propagated by the topic route by using the **unregister_instance()** operation. Default: true	0 or 1
<entity_monitoring>	Configures remote monitoring for the topic route. See Monitoring (Section 2.4.4) and Chapter 5: Monitoring Federation Service from a Remote Location.	0 or 1

2.4.6.1 Topic Route Types

The tag **<registered_type_name>** within the **<input>** and **<output>** tags contains the registered type name of the topic. The actual definition of that type can be set in the configuration file (see Section 2.4.6.2) or it can be discovered from an entity on the domain route's domains (see Section 2.4.6.3).

Table 2.10 **Input and Output Tags for a Topic Route**

Tags within <topic_route> <input>	Tags within <topic_route> <output>	Description	Number of Tags Allowed
<registered_type_name>		Sets the registered type name of this topic. See Topic Route Types (Section 2.4.6.1).	1 ***(required)***
<topic_name>		Sets the topic name.	1 ***(required)***
<creation_mode>		Specifies when to create the DataReader/DataWriter. Default: IMMEDIATE See Creation Modes—Controlling When a Topic Route is Created (Section 2.4.6.4).	0 or 1

<content_filter>	N/A	Defines a SQL content filter for the DataReader.
		Example:
		`<topic_route>`
		`...`
		`<input>`
		`...`
		`<content_filter>`
		`<expression>`
		`x > 100`
		`</expression>`
		`</content_filter>`
		`...`
		`</input>`
		`...`
		`</topic_route>`
			0 or 1

<datareader_qos>	<datawriter_qos>	Sets the DataReader or DataWriter QoS. The contents of this tag are specified in the same manner as a *Connext* QoS profile file—see Chapter 15 in the *RTI Core Libraries and Utilities User's Manual*. If the tag is not defined, *Federation Service* will use the *Connext* defaults.	0 or 1

2.4.6.2 Defining Types in the Configuration File

To define and use a type in your XML configuration file:

1. Define your type within the **<types>** tag. (This is one of the top-level tags, see Table 2.1.)
2. Register it in the participant(s) where you will use it.
3. Refer to it in the domain route(s) that will use it.

For example:

```
<dds>
    ...
    <types>
        <struct name="PointType">
        ...
        </struct>
    </types>
    ...
    <federation_service name=”MyFederationService”>
        ...
        <domain_route>
            <participant_1>
                ...
            </participant_1>
            <participant_2>
                ...
                <registered_type name="Position" type_name="PointType"/>
            </participant_2>
            ...
            <session>
                <topic_route>
                    <input participant="2">
                        <registered_type_name>Position
                        </registered_type_name>
                    </input>
                    <output>
                        ...
                    </output>
                </topic_route>
            </session>
            ...
        </domain_route>
        ...
    </federation_service>
    ...
<dds>
```

The type description is done using the *Connext* XML format for type definitions. For more information, see Section 3.4 in the *RTI Core Libraries and Utilities User's Manual*.

2.4.6.3 Discovering Types

If a topic route refers to types that are not defined in the configuration file, the federation service has to discover their typecodes. A topic route cannot be created without the typecode information.

By default, the DataReader creation will be tied to the discovery of types (typecodes) in the input domain and the DataWriter creation will be tied to the discovery of types (typecodes) in the output domain. If you want to use types discovered in either one of the domains for the creation of both the DataReader and DataWriter, you must set the **<route_types>** tag to true.

In the following example, both the DataWriter and DataReader will be created as soon as the type **ShapeType** is discovered in either domain.

```
<topic_route>
    <route_types>true</route_types>
    <input participant="1">
        <creation_mode>IMMEDIATE</creation_mode>
        <registered_type_name>ShapeType</registered_type_name>
        ...
    </input>
    <output>
        <creation_mode>IMMEDIATE</creation_mode>
        <registered_type_name>ShapeType</registered_type_name>
        ...
    </output>
    ...
</topic_route>
```

In this next example, the DataReader will be created only when the type **ShapeType** is discovered in the input domain; the DataWriter will be created only when the type **ShapeType** is discovered in the output domain.

```
<topic_route>
    <route_types>false</route_types>
    <input participant="1">
        <creation_mode>IMMEDIATE</creation_mode>
        <registered_type_name>ShapeType</registered_type_name>
        ...
    </input>
    <output>
        <creation_mode>IMMEDIATE</creation_mode>
```

```
            <registered_type_name>ShapeType</registered_type_name>
            ...
        </output>
        ...
</topic_route>
```

2.4.6.4 Creation Modes—Controlling When a Topic Route is Created

The way a topic route creates its DataReader and DataWriter and starts reading and writing data can be configured.

The **<creation_mode>** tag in a topic route's **<input>** and **<output>** tags controls *when* the federation service DataReader/DataWriter is created. Table 2.11 lists the possible values for the **<creation_mode>** tag.

Table 2.11 **Creation Modes**

<creation_mode> Values	Description
IMMEDIATE *(default)*	The federation service DataReader/DataWriter is created as soon as possible; that is, as soon as the types are available. Note that if the type is defined in the configuration file, the creation will occur when the federation service starts. If the type is not defined in the configuration file, it has to be discovered; see Discovering Types (Section 2.4.6.3).
ON_DOMAIN_MATCH	The federation service DataReader is not created until the federation service discovers a user DataWriter on the same topic in its domain. The federation service DataWriter is not created until the federation service discovers a user DataReader on the same topic in its domain.
ON_ROUTE_MATCH	The federation service DataReader/DataWriter is not created until its counterpart in the topic route is created.
ON_DOMAIN_AND_ROUTE_MATCH	Both conditions must be true.
ON_DOMAIN_OR_ROUTE_MATCH	At least one of the conditions must be true.

Topic Route Destruction:

The same rules that are applied to create the federation service DataWriter and DataReader also apply to their destruction. When the condition that triggered the creation of that entity becomes false, the entity is destroyed. (Note that IMMEDIATE will never become false.)

For example, if the creation mode of a topic route's **<input>** tag is ON_DOMAIN_MATCH, when all the matching user DataWriters are deleted, the topic route's DataReader is deleted.

Example 1

In this example, data is routed as soon as a user DataWriter is publishing it on the first domain.

```
<topic_route>
    <input participant="1">
        <creation_mode>ON_DOMAIN_MATCH</creation_mode>
        ...
    </input>
    <output>
        <creation_mode>ON_ROUTE_MATCH</creation_mode>
    ...
    </output>
</topic_route>
```

Example 2

In this example, data is not routed until a user DataWriter is publishing and a user DataReader is already expecting it.

```
<topic_route>
    <input participant="1">
        <creation_mode>ON_DOMAIN_AND_ROUTE_MATCH</creation_mode>
        ...
    </input>
    <output>
        <creation_mode>ON_DOMAIN_AND_ROUTE_MATCH</creation_mode>
        ...
  </output>
</topic_route>
```

Example 3

In this example, all the data is received by the topic route's DataReader, because it is created as soon as a user DataWriter is discovered on the first domain. However, the data is not resent until a user DataReader on the other domain subscribes to it.

```
<topic_route>
    <input participant="1">
        <creation_mode>ON_DOMAIN_MATCH</creation_mode>
        ...
    </input>
    <output>
```

```
        <creation_mode>ON_DOMAIN_AND_ROUTE_MATCH</creation_mode>
        ...
    </output>
</topic_route>
```

2.4.7 Auto Topic Routes

The tag **<auto_topic_route>** defines a set of potential topic routes, with the same input and output type, and the same input and output topic. A topic route can eventually be instantiated when the type and topic of a discovered entity match the filters in the auto topic route. When this happens, a topic route is created (but not necessarily started; see Section 2.4.6.4) with the configuration defined in the auto topic route.

For example:

```
<dds>
    ...
    <federation_service>
        ...
        <domain_route>
            ...
            <session>
                ...
                <auto_topic_route name="AutoRoute1">
                    ...
                    <input participant="1">
                        <allow_topic_name_filter>*
                        </allow_topic_name_filter>
                        <allow_registered_type_name_filter>
                            ShapeType
                        </allow_registered_type_name_filter>
                        ...
                    </input>
                    <output>
                        <allow_topic_name_filter>A*
                        </allow_topic_name_filter>
                        <allow_registered_type_name_filter>
                            B*
                        </allow_registered_type_name_filter>
                        ...
                    </output>
                </auto_topic_route>
                ...
            </session>
            ...
```

```
        </domain_route>
        ...
    </federation_service>
    ...
</dds>
```

The previous auto topic route will lead to the creation of a topic route every time any topic of type ShapeType is discovered on the first participant or a topic starting with A with a type starting with B is discovered on the second participant.

For example, discovering the topic "Triangle" of "ShapeType" will trigger the creation of a topic route that routes triangles from the first to the second domain. Discovering a topic "Atopic" of type "Btype" on the second domain will trigger the creation of a topic route that routes "Atopic" from the first to the second domain too.

Table 2.12 lists the tags allowed within an **<auto_topic_route>** tag.

Table 2.13 lists the tags allowed within **<input>** and **<output>** tags nested within an **<auto_topic_route>** tag.

Table 2.12 **Auto-topic Route Tags**

Tag within <auto_topic_route>	Description	Number of Tags Allowed
<input>	Defines the auto topic route's input topic. See Auto Topic Routes (Section 2.4.7).	1 ***(required)***
<output>	Defines the auto topic route's output topic. See Auto Topic Routes (Section 2.4.7).	1 ***(required)***
<publish_with_original_info>	The topic routes are created with this configuration. When this flag is set to true, if you have N topic routes for the same topic (in different routers or in the same one), each sample that was written from a DataWriter in the input domain will be routed N times, but DataReaders on the output domain will only see one. Default: false	0 or 1
<publish_with_original_timestamp>	The topic routes are created with this configuration. When this tag is set to true, the data samples are written with their original source timestamp. Default: false	0 or 1

Table 2.12 **Auto-topic Route Tags**

Tag within <auto_topic_route>	Description	Number of Tags Allowed
<propagate_dispose>	The topic routes are created with this configuration. When this flag is set to true, the NOT_ALIVE_DISPOSE samples received by the topic routes' DataReaders are not published with the topic routes' DataWriters. Default: true	0 or 1
<propagate_unregister>	The topic routes are created with this configuration. When this flag is set to true, the NOT_ALIVE_NO_WRITERS samples received by the topic routes' DataReaders are not published with the topic routes' DataWriters as unregister messages. Default: true	0 or 1
<monitoring>	Enables and configures remote monitoring for the auto topic route. See Monitoring (Section 2.4.4) and Chapter 5: Monitoring Federation Service from a Remote Location	0 or 1

Table 2.13 **Input and Output Tags for the <auto_topic_route> Tag**

Tags within <input>	Tags within <output>	Description	Number of Tags Allowed
<creation_mode>		The topic routes are created with this configuration. See Creation Modes—Controlling When a Topic Route is Created (Section 2.4.6.4).	0 or 1

Table 2.13 **Input and Output Tags for the <auto_topic_route> Tag**

Tags within <input>	Tags within <output>	Description	Number of Tags Allowed

<content_filter>	N/A	The topic routes are created with a SQL content filter topic with this expression.
		`<auto_topic_route>`
		`...`
		`<input>`
		`...`
		`<content_filter>`
		`<expression>`
		`x > 100`
		`</expression>`
		`</content_filter>`
		`...`
		`</input>`
		`...`
		`</auto_topic_route>`
			0 or 1

<datareader_qos>	<datawriter_qos>	The topic routes are created with this configuration. The contents of this tag are specified in the same manner as for a *Connext* QoS profile file—see Chapter 15 in the *RTI Core Libraries and Utilities User's Manual*. If the tag is not defined, *Federation Service* will use the *Connext* defaults.	0 or 1
<allow_topic_name_filter>		The topic name filter.[1] Default:* (allow all)	0 or 1
<allow_registered_type_name_filter>		The registered type name filter.[1] Default:* (allow all)	0 or 1
<deny_topic_name_filter>		A topic name filter[1] that should be denied (excluded). This is applied after the **<allow_topic_name_filter>**. Default: Not applied	0 or 1
<deny_registered_type_name_filter>		A registered type name filter[1] that should be denied (excluded). This is applied after the **<allow_registered_type_name_filter>**. Default: Not applied	0 or 1

1. As defined by the POSIX fnmatch API (1003.2-1992 section B.6).

2.5 Enabling and Disabling Federation Service Entities

The *Federation Service* entities associated with the tags **<federation_service>**, **<domain_route>**, **<topic_route>**, and **<auto_topic_route>** can be created enabled or disabled using the attribute **enabled**.

By default, the value of the **enabled** attribute is true.

For example:

```
<dds>
    <federation_service name="TopicBridgeExample"
     group_name="rti.router.default" enabled="true">
        <domain_route name="DomainRoute" enabled ="false">
            <participant_1>
                <domain_id>0</domain_id>
            </participant_1>
            <participant_2>
                <domain_id>1</domain_id>
            </participant_2>
            <session name="Session">
                <topic_route name="SquaresRoute" enabled="false">
                    <input participant="1">
                        <registered_type_name>
                            ShapeType
                        </registered_type_name>
                        <topic_name>Square</topic_name>
                    </input>
                    <output>
                        <registered_type_name>
                            ShapeType
                        </registered_type_name>
                        <topic_name>Square</topic_name>
                    </output>
                </topic_route>
            </session>
        </domain_route>
    </federation_service>
</dds>
```

When an entity is created disabled, it can be enabled remotely using the commands enable (Section 4.2.5) and disable (Section 4.2.4).

A **federation_service** can be created disabled by setting the attribute **enabled** to false or by using the **-noAutoEnable** command-line option. The command-line parameter takes precedence over the XML attribute value.

Chapter 3 Running Federation Service

This chapter describes:

- ❏ Starting Federation Service (Section 3.1)
- ❏ Stopping Federation Service (Section 3.2)

3.1 Starting Federation Service

Federation Service runs as a separate application. The script to run the executable is located in **<*Federation Service installation directory*>/scripts**.

To start Federation Service, enter:

```
cd <installation directory for Federation Service>
scripts/rtifederationservice [options]
```

Example:

```
cd <installation directory for Federation Service>
scripts/rtifederationservice \
    -cfgFile example/shapes/topic_bridge.xml -cfgName example
```

Table 3.1 describes the command-line options.

3.2 Stopping Federation Service

To stop *Federation Service*, press **Ctrl-c**. *Federation Service* will perform a clean shutdown.

Table 3.1 **Federation Service Command-line Options**

Option	Description
`-appName <name>`	Assigns a name to the execution of *Federation Service*. Remote commands and status information will refer to the service using this name. See the *Federation Service User's Manual* for more information. In addition, the name of *DomainParticipants* created by *Federation Service* will be based on this name. Default: The name given with **-cfgName** if present, otherwise it is "RTI_Federation_Service".
`-cfgName`	This name is used to find the matching **<federation_service>** tag in the configuration file. It is required unless both **-remoteAdministrationDomainId** and **-noAutoStart** are used, in which case no configuration is loaded at start-up and *Federation Service* will simply wait to be configured remotely.
`-domainIdBase <ID>`	Sets the base domain ID. This value is added to the domain IDs in the configuration file. For example, if you set **-domainIdBase** to 50 and use domain IDs 0 and 1 in the configuration file, then the *Federation Service* will use domains 50 and 51. Default: 0
`-help`	Displays help information.
`-licenseFile <file>`	Specifies the license file (path and filename). Only applicable to licensed versions of *Federation Service*. If not specified, *Federation Service* looks for the license as described in Installing the License File (Section 2.3) in the Getting Started Guide.
`-identifyExecution`	Appends the host name and process ID to the service name provided with the **-appName** option. This helps ensure unique names for remote administration and monitoring. For example: MyFederationService_myhost_20024
`-maxObjectsPerThread <int>`	Parameter for the DomainParticipantFactory.
`-noAutoEnable`	Starts *Federation Service* in a disabled state. Use this option if you plan to enable *Federation Service* remotely, as described in the *Federation Service User's Manual*. This option overwrites the value of the **enable** attribute in the **<federation_service>** tag.

Table 3.1 **Federation Service Command-line Options**

Option	Description
`-remoteAdministrationDomainId <ID>`	Enables remote administration and sets the domain ID for remote communication. When remote administration is enabled, *Federation Service* will create a DomainParticipant, Publisher, Subscriber, DataWriter, and DataReader in the designated domain. The QoS values for these entities are described in the *Federation Service User's Manual*. This option overwrites the value of the tag **<domain_id>** within a **<administration>** tag. (See the *Federation Service User's Manual* for information on configuring remote access). Default: remote administration is not enabled unless it is enabled from the XML file.
`-remoteMonitoringDomainId <ID>`	Enables remote monitoring and sets the domain ID for status publication. When remote monitoring is enabled, *Federation Service* will create one *DomainParticipant*, one *Publisher*, five *DataWriters* for data publication (one for each kind of entity), and five *DataWriters* for status publication (one for each kind of entity). The QoS values for these entities are described in the *Federation Service User's Manual*. This option overwrites the value of the tag **<domain_id>** within a **<monitoring>** tag. (See the *Federation Service User's Manual* for information on configuring remote monitoring). Default: remote monitoring is not enabled unless it is enabled from the XML file.
`-sleep <seconds>`	Sleeps for x seconds before executing the next command. This is useful when issuing commands from a command file (see **-cmdName**).
`-stopAfter <sec>`	Stops the service after the specified number of seconds.
`-use42eAlignment`	Enables compatibility with *RTI Data Distribution Service* 4.2e. This option should be used when compatibility with 4.2e is required and the topic data types contain double, long long, unsigned long long, or long double members. Default: disabled

Table 3.1 **Federation Service Command-line Options**

Option	Description
`-verbosity <n>`	Controls what type of messages are logged: 0 - Silent 1 - Exceptions (Core Libraries and *Federation Service*) (default) 2 - Warnings(*Federation Service*) 3 - Information (*Federation Service*) 4 - Warnings (Core Libraries and *Federation Service*) 5 - Tracing (*Federation Service*) 6 - Tracing (Core Libraries and *Federation Service*) Each verbosity level, *n*, includes all the verbosity levels smaller than *n*.
`-version`	Prints the *Federation Service* version number.

Chapter 4 Administering Federation Service from a Remote Location

Federation Service can be controlled remotely by sending commands through a special Topic. Any *Connext* application can be implemented to send these commands and receive the corresponding responses. A shell application that sends/receives these commands is provided with *Federation Service*.

The script for the shell application is in ***<Federation Service installation directory>/scripts/rtifssh***.

Entering **rtifssh -help** will show you the command-line options:

```
RTI Federation Service Shell
Usage: rtifssh [options]...
Options:
 -domainId  <integer>   Domain id for the remote configuration
 -timeout   <seconds>   Max time to wait a remote response
 -cmdFile   <file>      Run commands in this file
 -help                  Displays this information
```

4.1 Enabling Remote Administration

By default, remote administration is disabled in *Federation Service* for security reasons.

To enable remote administration you can use the **<administration>** tag (see Section 2.4.3) or the **-remoteAdministrationDomainId <ID>** command-line parameter, which enables remote administration and sets the domain ID for remote communication. For more information about the command-line options, see Section 3.1 in the Getting Started Guide.

When remote administration is enabled, *Federation Service* will create a DomainParticipant, Publisher, Subscriber, DataWriter, and DataReader in the designated domain. (The QoS values for these entities are described in Section 2.4.3.)

4.2 Remote Commands

This section describes the remote commands using the shell interface; Section 4.3 explains how to use remote administration from a *Connext* application.

Remote commands:

- add_peer <*target_federation_service*> <*domain_route_name*> p1 | p2 <*peer_list*>
- create <*target_federation_service*>
  domain_route | session | topic_route | auto_route [<*parent_entity_name*>]
  <*xml_url*> [remote | local]
- delete <*target_federation_service*> [<*entity_name*>]
- disable <*target_federation_service*> [<*entity_name*>]
- enable <*target_federation_service*> [<*entity_name*>]
- load <*target_federation_service*> <*cfg_name*><*xml_url*> [remote | local]
- pause <*target_federation_service*> [<*entity_name*>]
- resume <*target_federation_service*> [<*entity_name*>]
- save <*target_federation_service*>
- enable <*target_federation_service*>
- update <*target_federation_service*> [<*entity_name*>] [<*xml_url*> | <*assignment_expr*>]
  [remote | local]

Parameters:

❑ <*assignment_expr*> can be used instead of <xml_url> to modify single values in an entity configuration.

The assignment expression has the form:

<*fully qualified value name*> = <*value*>

For example:

```
update ShapeRouter DomainRoute1::Session1::SquareRoute
  topic_route.input.datareader_qos.deadline.period.sec = 3
```

```
update ShapeRouter DomainRoute1::Session1::SquareRoute
  topic_route.input.content_filter.expression = "x < 30"
```

- ❑ <*domain_route_name*> is the fully qualified name of a domain route entity
- ❑ <*entity_name*> is a fully qualified name. For example, consider the following XML configuration:

```
<federation_service name="ShapeFederationService">
    ...
    <domain_route name="DomainRoute1">
        ...
        <session name="Session">
            <topic_route name="SquaresRoute">
                ...
```

The above XML configuration would allow you to use commands such as:

- `enable ShapeFederationService DomainRoute1::Session::SquaresRoute`
- `enable ShapeFederationService DomainRoute1`

Note that the fully qualified name does not include the name of the federation service.

- ❑ <*peer_list*> is a comma-separate list of peers
- ❑ <*target_federation_service*> can be:
  - The application name of a federation service, such as "**MyFederationService1**", as specified at start-up with the command-line option **-appName**
  - A regular expression[1] for a federation service name, such as **"MyFederationService*"**
- ❑ <*xml_url*> can be:
  - A file URL, such as **file:///home/user/myconfig.xml**
  - A string URL, such as:

```
str://"<topic_route><input><datareader_qos>...
</datareader_qos></input></topic_route>"
```

If you omit the URL schema name, *Federation Service* will assume a file name; for example, **/home/user/myconfig.xml** is equivalent to **file:///home/user/myconfig.xml**.

1. As defined by the POSIX fnmatch API (1003.2-1992 section B.6)

In either case, the XML code can represent either a whole DTD-valid configuration file (such as the one you specify through the command-line when you start *Federation Service*) or a snippet of XML that only refers to a specific entity (this is further explained in Section 4.2.11).

The **[remote|local]** parameter is used with file URLs to indicate if the file is local to the shell (local) or local to the federation service (remote). If the file is local to the shell (local), the shell application will read it and will send it as a string URL. If the file is local to the federation service (remote), the shell will send it as a file URL that will be read by the federation service. The default value is **remote**.

If a relative path is specified, the path will be relative to the working directory in which the federation service (if **remote** is specified) or shell (if **local** is specified) is running.

4.2.1 add_peer

```
add_peer <target_federation_service> <domain_route_name> p1|p2
   <peer_list>
```

The **add_peer** command passes the *peer_list* to the underlying DomainParticipant's **add_peer()** function. It is only valid for participants in a domain route.

<*domain_route_name*> is like <*entity_name*>, but must be a domain route entity.

p1|p2 specifies if the DomainParticipant associated with <participant_1> or <participant_2> configuration is selected.

<*peer_list*> is a comma-separated list of peers.

4.2.2 create

```
create <target_federation_service>
   domain_route|session|topic_route|auto_route [<parent_entity_name>]
   <xml_url> [remote|local]
```

The **create** command is similar to update (Section 4.2.11), but the configuration is applied to a newly created entity instead of an existing one.

The second parameter (domain_route|session|topic_route|auto_route) is the kind of entity to be created. If the kind is a domain_route, there will be no parent. For the other kinds (session, topic_route, or auto_route), a <*parent_entity_name*> must be specified.

<*xml_url*> and [remote|local] are the same as used in update (Section 4.2.11), except that only XML snippets matching the entity kind are allowed. A full file (starting with <dds>...) is not valid.

For example (this would be entered as a single command, with no line-breaks):

```
create example topic_route DomainRoute::Session
str://"<topic_route name="TrianglesToTriangles">
<input participant="1"><registered_type_name>ShapeType
</registered_type_name><topic_name>Triangle</topic_name></input>
<output><registered_type_name>ShapeType</registered_type_name>
<topic_name>Triangle</topic_name></output></topic_route>"
```

4.2.3 delete

delete <target_federation_service> [<entity_name>]

You can invoke the **delete** command on domain routes, topic routes and auto topic routes. It acts like the disable (Section 4.2.4) command, but also purges the configuration data for the target entity.

For example:

```
delete example DomainRoute::Session::CirclesToCircles
```

A deleted entity cannot be re-enabled, but a new one can be created.

4.2.4 disable

disable <target_federation_service> [<entity_name>]

The **disable** command disables a federation service entity by destroying its sub-entities and corresponding *Connext* objects:

- ❏ Federation service—When a federation service is disabled, all of its domain routes are destroyed. You do not need to specify the *entity_name* to disable a federation service.
- ❏ Domain route—When a domain route is disabled, all its topic routes and auto topic routes are destroyed, as well as both DomainParticipants. All the session threads are stopped and their corresponding Publishers and Subscribers are also deleted.
- ❏ Route, topic route, and auto topic route—When a topic route or auto topic route is disabled, its DataReaders and DataWriters are destroyed, so data will no longer be routed.

4.2.5 enable

```
enable <target_federation_service> [<entity_name>]
```

The **enable** command enables an entity that has been disabled or marked as '**enabled=false**' in the configuration file.

This command can be used to enable the following entities:

- Federation service—When a federation service is enabled, it uses the currently loaded configuration and starts. You don't need to specify the *entity_name* to enable a federation service.
- Domain route—When a domain route is enabled, it creates the DomainParticipants, topic routes, and auto topic routes that it contains. The topic routes and auto topic routes will be created enabled or disabled depending on their current configuration. Enabling a domain route is required to start federation data from the input domain to the output domain.
- Topic route, and auto topic route—Enabling a topic route or auto topic route is a necessary condition to start sending data between input and output streams. However, data federation will not start until the DataWriter and DataReader associated with a topic route are created (see Section 2.4.6.4 for additional information).

4.2.6 load

```
load <target_federation_service> <cfg_name> <xml_url> [remote|local]
```

The **load** command loads specific XML configuration code. The *target_federation_service* must be disabled. For more information, see How to Load the XML Configuration (Section 2.2).

The XML code received must represent a valid federation service configuration file. The name of the <federation_service> tag to load is identified with <cfg_name>.

4.2.7 pause

```
pause <target_federation_service> <entity_name>
```

When the pause command is called in a route, the session thread containing this topic route will stop reading data from the topic route's DataReader.

For federation service, domain routes, and auto topic routes, the execution of this command will pause the contained topic routes.

4.2.8 resume

```
resume <target_federation_service> <entity_name>
```

When the resume command is called in a route, the session thread containing this route will continue reading data from the topic route's DataReader.

For federation service, domain routes, and auto topic routes, the execution of this command will resume the contained topic routes.

4.2.9 save

```
save <target_federation_service>
```

This command writes the current configuration to a file. The file itself is specified with <save_path> (see page 2-17). If <save_path> has not been specified, the **save** command will fail. If the file specified by <save_path> already exists, the file will be overwritten.

The saved configuration is *functionally* equivalent to the loaded XML file plus any updates (either from an **update** command or other remote commands that change the configuration, such as **add_peer**). However it may not be *textually* equivalent. For example, the saved XML configuration may explicitly contain default values that were not in the initial XML.

4.2.10 unload

```
unload <target_federation_service>
```

The **unload** command unloads the current configuration that the *target_federation_service* is using, so you can change it with a subsequent load (Section 4.2.6) command.

The *target_federation_service* must be disabled for this command to succeed.

4.2.11 update

```
update <target_federation_service> [<entity_name>]
   [<xml_url>|<assignment_expr> [remote|local]
```

The **update** command changes the configuration of a specific entity. Table 4.1 shows the parameters that can be changed for each entity.

If you try to change an immutable parameter in an entity that is enabled, you will receive an error message. To change an immutable parameter, you must disable the federation service entity, change the parameter, and then enable the federation service entity again.

Table 4.1 **Changeable Parameters**

Entity	Mutable (changeable any time)	Immutable (only changeable when entity is *disabled*)[1]
Federation Service	<monitoring>/<enabled> <monitoring>/<status_publication_period> <entity_monitoring>/<enabled> <entity_monitoring>/<status_publication_period> <administration>/<save_path> <administration>/<autosave_on_update>	<monitoring>/<statistics_sampling_period> <monitoring>/<historical_statistics> <monitoring>/<domain_id> <entity_monitoring>/<statistics_sampling_period> <entity_monitoring>/<historical_statistics> <administration>/ *<all except save_path and autosave_on_update>*
Domain route	<participant_x>: Mutable QoS policies in <participant_qos> <entity_monitoring>/<enabled> <entity_monitoring>/<status_publication_period>	<participant_qos>: Immutable QoS policies in <participant_qos> <entity_monitoring>/<statistics_sampling_period> <entity_monitoring>/<historical_statistics>
Session	Mutable QoS policies in <publisher_qos> and <subscriber_qos> <entity_monitoring>/<enabled> <entity_monitoring>/<status_publication_period>	Immutable QoS policies in <publisher_qos> and <subscriber_qos> <entity_monitoring>/<statistics_sampling_period> <entity_monitoring>/<historical_statistics>
Topic Route	Mutable QoS policies in <datawriter_qos> and <datareader_qos> <route_types> <propagate_dispose> <publish_with_original_timestamp> <propagate_unregister> <publish_with_original_info> <content_filter>/<parameter> <entity_monitoring>/<enabled> <entity_monitoring>/<status_publication_period>	Immutable QoS policies in <datawriter_qos> and <datareader_qos> <creation_mode> <content_filter>/<expression> <entity_monitoring>/<statistics_sampling_period> <entity_monitoring>/<historical_statistics>
Auto Topic Route	Mutable QoS policies in <datawriter_qos> and <datareader_qos> <propagate_dispose> <propagate_unregister> <publish_with_original_info> <publish_with_original_timestamp> <content_filter>/<parameter> <entity_monitoring>/<enabled> <entity_monitoring>/<status_publication_period>	Immutable QoS policies in <datawriter_qos> and <datareader_qos> <creation_mode> <allow_topic_name_filter> <allow_registered_type_name_filter> <deny_topic_name_filter> <deny_registered_type_name_filter> <content_filter>/<expression> <entity_monitoring>/<statistics_sampling_period> <entity_monitoring>/<historical_statistics>

1. Monitoring parameters can also be changed when monitoring is disabled (even when the entity is enabled).

You can send an XML snippet (or an assignment expression) that only contains the values you want to change for that entity, or you can send a whole well-formed configuration file.

- If you send an XML snippet (or an assignment expression), only the changes you specify will take effect.

  For example, suppose you send this command:

  ```
  update ShapeRouter DomainRoute1::Session1::SquareRoute
    str://"<topic_route><input><datareader_qos><deadline><period>
    <sec>1</sec></period></deadline></datareader_qos></input>
    </topic_route>"
  ```

  or

  ```
  update ShapeRouter DomainRoute1::Session1::SquareRoute
    topic_route.input.datareader_qos.deadline.period.sec = 1
  ```

  The topic route **DomainRoute1::Session1::SquareRoute** will only change the **period** value in the Deadline QoS for that particular DataReader.

  Now suppose that later on you send this command:

  ```
  update ShapeRouter DomainRoute1::Session1::SquareRoute
    str://"<topic_route><input><datareader_qos><property>
    <value><element><name>MyProp</name><value>MyValueRemote</value>
    </element></value></property><datareader_qos></input>
    </topic_route>"
  ```

  This would only change the Property QoS; the Deadline QoS would keep the setting from the prior command.

- If you send a well-formed configuration file (starting with **<dds><federation_service>**), the properties in the route (QoS values in the topic route) will be completely replaced with the properties (QoS values) defined in the XML code. If a QoS value for a topic route is not defined in the XML code, *Federation Service* will use the *Connext* default.

4.3 Accessing Federation Service from Connext Applications

You can send commands to control a *Federation Service* instance from your own *Connext* application. You will need to create a DataWriter for a specific topic and type. Then, you can send a sample that contains a command and its parameters. Optionally, you can create a DataReader for a specific topic to receive the results of the execution of your commands.

The topics are:

- **rti/federation_service/administration/command_request**
- **rti/federation_service/administration/command_response**

The types are:

- **RTI::FederationService::Administration::CommandRequest**
- **RTI::FederationService::Administration::CommandResponse**

You can find the IDL definitions for these types in ***<Federation Service installation directory>*/resource/idl/FederationServiceAdministration.idl**.

The QoS configuration of your DataWriter and DataReader must be compatible with the one used by the federation service (see how this QoS is configured in Section 2.4.3).

When you send an XML string URL (str://"<xml_code>") with the **load** and **update** commands, if the string is longer than XML_URL_MAX_LENGTH (in the IDL file), you will have to split the string and send several samples, setting the "**final**" field to false in all but the last sample.

The following example shows how to send a command to update the Deadline QoS policy for a topic route's DataReader:

```
/* Create entities: participant, publisher, topic, datawriter...*/
/* ... */
RTI_FederationService_CommandRequest * cmdRequest =
    RTI_FederationService_CommandRequestTypeSupport::create_data();

/* By specifying an unique ID for this command, you will be able
   to identify its response later on */
cmdRequest->id.host = /* host ID */;
cmdRequest->id.app =  /* process ID */;
cmdRequest->id.invocation = ++invocationCounter;

/* Send this command to a federation service called MyRouter */
strcpy(cmdRequest->target_router, "MyRouter");

/* The command type is update */
```

```
cmdRequest->command._d = RTI_FEDERATION_SERVICE_COMMAND_UPDATE;

/* Specify entity name to update and the XML code to define
   the new configuration */

strcpy(cmdRequest->command._u.entity_desc.name,
       "DomainRoute1::Session1::TopicRoute1");

/* When we use an XML snippet, the first tag we specify is
   that of the entity, <topic_route> in this case */

strcpy(cmdRequest->command._u.entity_desc.xml_url.content,
       "str://\"<topic_route>\
                <input>\
                  <datareader_qos>\
                    <deadline>\
                        <period>\
                            <sec>10</sec>\
                        </period>\
                    </deadline>\
                  </datareader_qos>\
                 </input>\
                </topic_route>\"");
/* The content above is small enough to send it in one sample.
   Otherwise (if the length were > XML_URL_MAX_LENGTH) we would have
   to split it in multiple partial strings, each < XML_URL_MAX_LENGTH,
   and set final = 0 for all the samples but the last one */

cmdRequest->command._u.entity_desc.xml_url.final = 1;

RTI_FederationService_CommandRequest_writer->write(cmdRequest, ...);
```

Chapter 5 Monitoring Federation Service from a Remote Location

You can monitor *Federation Service* remotely by subscribing to special topics. By subscribing to these topics, any *Connext* application can receive information about the configuration and operational status of *Federation Service*.

Being able to monitor the state of a *Federation Service* instance is an important tool that allows you to detect problems. For example, looking at the input samples per second in the different sessions, you might see that one session is receiving most of the traffic. In that case, you could reassign some of the routes to other sessions to improve load balancing.

Federation Service can publish status for the following kinds of entities:

1. *Federation Servicee* itself (<federation_service>)
2. Domain Route <domain_route>)
3. Session (<session>)
4. Topic Route (<topic_route>) Auto Topic Route (<auto_topic_route>)

For each of the above kinds of entities, *Federation Service* creates two topics:

- **rti/federation_service/monitoring/<*tag*>_data** describes the entity's *configuration*
- **rti/federation_service/monitoring/<*tag*>_status_set** describes the entity's *operational status*

With the corresponding types:

- **RTI::FederationService::Monitoring::<*tag*>Data**
- **RTI::FederationService::Monitoring::<*tag*>StatusSet**

Where <*tag*> is one of the following entity kind tags: **FederationService**, **DomainRoute**, or **Session**, **Route** or AutoRoute.

5.1 Enabling Remote Monitoring

By default, remote monitoring is disabled in *Federation Service* for security and performance reasons.

To enable remote monitoring, you can use the **<monitoring>** tag (see Section 2.4.4) or the **–remoteMonitoringDomainId** command-line parameter, which enables remote monitoring and sets the domain ID for data publication. For more information about the command-line options, see Section 3.1 in the Getting Started Guide.

When remote monitoring is enabled, *Federation Service* creates:

- ❏ 1 DomainParticipant
- ❏ 1 Publisher
- ❏ 5 DataWriters for publishing *configuration data* (one for each kind of entity)
- ❏ 5 DataWriters for publishing *status* (one for each kind of entity).

The QoS values for these entities are described in Section 2.4.4.

5.2 Monitoring Configuration Data

Configuration data for *Federation Service* entities is published in entity *data* topics. These topics are similar to the *Connext* builtin topics (DCPSParticipant, DCPSPublication, and DCPSSubscription) that provide information about the configuration of remote entities.

This configuration data is published when:

- ❏ An entity is created or enabled.
- ❏ An entity is disabled or destroyed (a dispose message is published).
- ❏ The entity's configuration is modified using the remote command "**update**" (see Section 5.2.11).

- The entity's configuration is modified due to certain events in *Federation Service*. For example, discovery events may trigger the creation of DataWriters and DataReaders in a route.

The following sections describe the data available for each kind of *Federation Service* entity.

- Configuration Data for the Federation Service (Section 5.2.1)
- Configuration Data for a Domain Route (Section 5.2.2)
- Configuration Data for a Session (Section 5.2.3)
- Configuration Data for a Topic Route (Section 5.2.4)
- Configuration Data for an Auto Topic Route (Section 5.2.5)

Each section describes the IDL for the topics' underlying data types. The IDL is also in the file ***<Federation Service installation directory>*/resource/idl/FederationService-Monitoring.idl**.

5.2.1 Configuration Data for the Federation Service

The topic that publishes configuration data is called **rti/federation_service/monitoring/federation_service_data**. This topic describes the configuration of the federation service but not its contained entities.

The IDL definition of the data type is:

```
struct FederationServiceAdministrationData {
    string<EXPRESSION_MAX_LENGTH> save_path;
    boolean autosave_on_update;
};
struct FederationServiceData {
    string<ENTITY_NAME_MAX_LENGTH> name; //@key
    string<ENTITY_NAME_MAX_LENGTH> group_name;
    string<ENTITY_NAME_MAX_LENGTH> host_name;
    long host_id;
    long app_id;
    FederationServiceAdministrationData administration;
};
```

Table 5.1 on page 5-4 describes the members of the FederationServiceData data type.

Table 5.1 **FederationServiceData**

Field Name	Key	Description
name	✓	Name of the federation service instance. The name associated with the federation service instance can be assigned explicitly using the command-line parameter **–appName**. If **–appName** is not used, the **<federation_service>** tag name provided with **-cfgName** is used. If you use **-identifyExecution**, the host name and process ID are appended to the name. For example: **RTI_FederationService_myhost_1234**
group_name		Name of the group to which the federation service belongs. Federation services in the same group will not communicate with each other. The group name is assigned using the attribute **group_name** in the **<federation_service>** tag. If the attribute is not defined, the group name is automatically set to **RTI_FederationService_<*Host Name*>_<*Process ID*>**
host_name		Name of the host where the federation service is running.
host_id		Identifies the host where the federation service instance is running.
app_id		Process (task) ID of the federation service instance.
administration. save_path		Specifies the file that will contain the saved configuration.
administration. auto_save_on_update		A boolean that, if true, automatically triggers a **save** command when configuration updates are received.

Federation service data samples are published when:

- ❏ The federation service instance is enabled.
- ❏ The federation service instance is disabled (dispose sample).
- ❏ Monitoring is enabled via remote administration.

5.2.2 Configuration Data for a Domain Route

The topic that publishes domain route configuration data is called **rti/federation_service/monitoring/domain_route_data**. The domain route data describes the configuration of the domain route and its participants but not its contained entities.

The IDL definition of the data type RTI::FederationService::Monitoring::DomainRouteData is:

```
struct DomainRouteParticipantData {
    long domain_id;
    BuiltinTopicKey_t participant_key;
};
struct DomainRouteData {
    string<ENTITY_NAME_MAX_LENGTH> federation_service_name; //@key
    string<ENTITY_NAME_MAX_LENGTH> name; //@key
    DomainRouteParticipantData participant_1;
    DomainRouteParticipantData participant_2;
};
```

Table 5.2 describes the members of the DomainRouteData data type.

Table 5.2 **DomainRouteData**

Field Name	Key	Description
federation_service_name	✓	The federation service name (assigned using **–appName**).
name	✓	The domain route name. This is configured using the **name** attribute in the **<domain_route>** tag.
participant_1		The configuration of the <participant_1>.
participant_2		The configuration of the <participant_2>.
participant_1.domain_id		Domain ID of the first domain route participant.The domain ID is configured using the XML tag <domain_id> inside <participant_1>.
participant_2.domain_id		Domain ID of the second domain route participant.The domain ID is configured using the XML tag <domain_id> inside <participant_2>.
participant_1.participant_key		Unique identifier for the first DomainParticipant.
participant_2.participant_key		Unique identifier for the second DomainParticipant.

A domain route can be correlated with its corresponding DomainParticipants using the fields participant_1.participant_key and/or participant_2.participant_key.

For example, let's assume that we want to get the value of the PropertyQosPolicy associated with the first DomainParticipant of a domain route. To do that, we would subscribe to the participant builtin topic and look for a sample where the key member is equal to

participant_1.participant_key. From this sample, we can get the PropertyQosPolicy by accessing the member called **property**.

For additional information on how to subscribe to builtin topics, see the *RTI Core Libraries and Utilities User's Manual*.

Domain-route data samples are published when:

- ❏ The domain route is enabled.
- ❏ The domain route is disabled (dispose sample).
- ❏ Monitoring is enabled via remote administration.

5.2.3 Configuration Data for a Session

The topic that publishes session configuration data is called **rti/federation_service/monitoring/session_data**. The session data describes the configuration of the session but not its contained entities.

The IDL definition of the data type is:

```
struct SessionData {
    string<ENTITY_NAME_MAX_LENGTH> federation_service_name; //@key
    string<ENTITY_NAME_MAX_LENGTH> domain_route_name; //@key
    string<ENTITY_NAME_MAX_LENGTH> name; //@key

    long enabled_route_count;
};
```

Table 5.3 describes the fields in the SessionData data type.

Table 5.3 **SessionData**

Field Name	Key	Description
federation_service_name	✓	The federation service name (assigned using **–appName**).
domain_route_name	✓	The domain route name.
name	✓	The session name, which is configured with the **name** attribute in the **<session>** tag.
enabled_route_count		The number of enabled routes.

Session data samples are published when:

- ❏ The session is enabled.
- ❏ The session is disabled (dispose sample).
- ❏ Atopic route inside the session is enabled.
- ❏ An auto_topic_route inside the session is disabled.
- ❏ Monitoring is enabled via remote administration.

5.2.4 Configuration Data for a Topic Route

The topic that publishes route configuration data is called **rti/federation_service/monitoring/route_data**.

The IDL definition of the data type RTI::FederationService::Monitoring::RouteData is:

```
struct RouteDdsInputData {
    long domain_id;
    BuiltinTopicKey_t datareader_key;
    string<EXPRESSION_MAX_LENGTH> content_filter_expression;
};

struct RouteDdsOutputData {
    long domain_id;
    BuiltinTopicKey_t datawriter_key;
};

struct RouteInputData {
    string<TOPIC_NAME_MAX_LENGTH> topic_name;
    string<TYPE_NAME_MAX_LENGTH> registered_type_name;

    long participant;RouteCreationMode creation_mode;
    DDSEntityState state;
    RouteDdsInputData dds_data;
};

struct RouteOutputData {
    string<TOPIC_NAME_MAX_LENGTH> topic_name;
    string<TYPE_NAME_MAX_LENGTH> registered_type_name;
    RouteCreationMode creation_mode;
    DDSEntityState state;
    RouteDdsOutputData dds_data;
};
```

```
struct RouteData {
   string<ENTITY_NAME_MAX_LENGTH> federation_service_name;//@key
   string<ENTITY_NAME_MAX_LENGTH> domain_route_name; //@key
   string<ENTITY_NAME_MAX_LENGTH> session_name; //@key
   string<ENTITY_NAME_MAX_LENGTH> name; //@key

   string<ENTITY_NAME_MAX_LENGTH> auto_route_name;

   boolean propagate_dispose;
   boolean propagate_unregister;
   boolean publish_with_original_info;
   boolean publish_with_original_timestamp;

   boolean route_types;

   RouteInputData input;
   RouteOutputData output;
};
```

Table 5.4 describes the fields in the RouteData topic data type.

Table 5.4 **RouteData**

Field Name	Key	Description
federation_service_name	✓	The federation service name (assigned with **–appName**).
domain_route_name	✓	The domain route name.
session_name	✓	The session name.
name	✓	The topic route name, which is configured using the **name** attribute in the <topic_route> tag.
auto_route_name		If the topic route is contained in an auto topic route, this field contains the auto topic route name. Otherwise, the field is initialized with the empty string.
input		The configuration of the topic route's input, as contained in the tag <input> inside <topic_route>
output		The configuration of the topic route's output, as contained in the tag <output> inside <topic_route>
input.participant		Index of the input participant (1 or 2). The value of this field is used to determine whether the input of this route is the domain route's participant 1 or participant 2.

Table 5.4 **RouteData**

Field Name	Key	Description
input.topic_name		Input topic name. The input topic name is configured using the tag <topic_name> inside <topic_route>/<input>
input.registered_type_name		Input registered name. The input registered name is configured using the tag <registered_type_name> inside <topic_route>/<input>.
output.topic_name		Output topic name. The output topic name is configured using the tag <topic_name> inside <topic_route>/<output>.
output.registered_type_name		Output registered name. The output registered name is configured using the tag <registered_type_name> inside <topic_route>/<output>.
input.creation_mode		Indicates when the DataReader is created in the input. The input creation mode is configured using the tag <creation_mode>.
output.creation_mode		Indicates when the DataWriter in created in the output. The output creation mode is configured using the tag <creation_mode>.
input.dds_data		Contains the configuration of the topic route's input.
output.dds_data		Contains the configuration of the topic route's output.
input.dds_data.domain_id		Domain ID of the input participant
output.dds_data.domain_id		Domain ID of the output participant
input.dds_data.datareader_key		Unique identifier for the DataReader. The value of this field is meaningful only when the state is RTI_FEDERATION_SERVICE_CREATED_AND_ENABLED.
output.dds_data.datawriter_key		Unique identifier for the DataWriter. The value of this field is only meaningful when datawriter_state is RTI_FEDERATION_SERVICE_CREATED_AND_ENABLED.
input.dds_data.content_filter_expression		Content filter expression associated with the content filter for the topic route DataReader. The expression is configured using the tag <content_filter>/<expression> inside <topic_route>/<input>.

Table 5.4 **RouteData**

Field Name	Key	Description
propagate_dispose		Indicates if the topic route propagates NOT_ALIVE_DISPOSE samples. The propagation of NOT_ALIVE_DISPOSE samples is configured using the tag <propagate_dispose> in <topic_route>.
propagate_unregister		Indicates if the topic route propagates NOT_ALIVE_NO_WRITERS samples. The propagation of NOT_ALIVE_NO_WRITERS samples is configured using the tag <propagate_unregister> in <topic_route>.
publish_with_original_info		Indicates if the topic route publishes the samples with original writer info. Setting this option to true allows redundant topic routes and prevents the applications from receiving duplicate samples. The publication with original writer info is configured using the tag <publish_with_original_info> inside <topic_route>.
publish_with_original_timestamp		Indicates if the topic route is configured to publish the output samples with the same timestamp as that of the input sample.
route_types		Indicates if the input participant will use types discovered in the output participant and vice versa for the creation of DataWriters and DataReaders. The route types flag is configured using the tag <route_types> inside <route> or <topic_route>.
input.state		Indicates whether or not the DataReader associated with a topic route is created.
output.state		Indicates whether or not the DataWriter associated with a topic route is created.

The correlation between a topic route and its DataReader and DataWriter can be done using the fields **datareader_key** and **datawriter_key**.

For example, let's assume that we want to retrieve the value of the DurabilityQosPolicy associated with the route's DataWriter. To do that, we would subscribe to the publication builtin topic and we would look for a sample where the key member is equal to **datawriter_key**. From this sample, we can get the DurabilityQosPolicy value accessing the member durability.

For additional information on how to subscribe to the builtin topics, see the *RTI Core Libraries and Utilities User's Manual*.

Route data samples are published when:

- ❑ The topic route is enabled.
- ❑ The topic route is disabled (dispose sample).
- ❑ The topic route configuration is modified using the remote command update.
- ❑ The topic route's DataReader is created.
- ❑ The topic route's DataReader is destroyed.
- ❑ The topic route's DataWriter is created.
- ❑ The topic route's DataWriter is destroyed.
- ❑ Monitoring is enabled via remote administration.

5.2.5 Configuration Data for an Auto Topic Route

The topic that publishes auto topic route configuration data is called **rti/federation_service/monitoring/auto_route_data**.

The IDL definition of the data type RTI::FederationService::Monitoring::AutoRouteData is:

```
struct AutoRouteDdsInputData {
    long domain_id;
    string<EXPRESSION_MAX_LENGTH> content_filter_expression;
};

struct AutoRouteDdsOutputData {
    long domain_id;
};

struct AutoRouteInputData {
    string<TOPIC_NAME_MAX_LENGTH> allow_topic_name_filter;
    string<TYPE_NAME_MAX_LENGTH> allow_registered_type_name_filter;
    string<TOPIC_NAME_MAX_LENGTH> deny_topic_name_filter;
    string<TYPE_NAME_MAX_LENGTH> deny_registered_type_name_filter;
    long participant;
    RouteCreationMode creation_mode;
    AutoRouteDdsInputData dds_data;
};
```

```
struct AutoRouteOutputData {
    string<TOPIC_NAME_MAX_LENGTH> allow_topic_name_filter;
    string<TYPE_NAME_MAX_LENGTH> allow_registered_type_name_filter;
    string<TOPIC_NAME_MAX_LENGTH> deny_topic_name_filter;
    string<TYPE_NAME_MAX_LENGTH> deny_registered_type_name_filter;

    RouteCreationMode creation_mode;
    AutoRouteDdsOutputData dds_data;
};

struct AutoRouteData {
    string<ENTITY_NAME_MAX_LENGTH> federation_service_name; //@key
    string<ENTITY_NAME_MAX_LENGTH> domain_route_name; //@key
    string<ENTITY_NAME_MAX_LENGTH> session_name; //@key
    string<ENTITY_NAME_MAX_LENGTH> name; //@key
    boolean propagate_dispose;
    boolean propagate_unregister;
    boolean publish_with_original_info;
    boolean publish_with_original_timestamp;
    long enabled_route_count;
    AutoRouteInputData input;
    AutoRouteOutputData output;
};
```

Table 5.5 describes the fields in the AutoRouteData data type.

Auto topic route data samples are published when:

- ❏ The auto topic route is enabled.
- ❏ The auto topic route is disabled (dispose sample).
- ❏ The auto topic route configuration is modified using the remote command update.
- ❏ A new route is created from the auto topic route
- ❏ Monitoring is enabled via remote administration.

Table 5.5 **AutoRouteData**

Field Name	Key	Description
federation_service_name	✓	The federation service name (assigned using **–appName**).
domain_route_name	✓	The domain route name.
session_name	✓	The session name.

Table 5.5 **AutoRouteData**

Field Name	Key	Description
name	✓	The auto topic route name, which is configured using the name attribute in the <auto_topic_route> tag.
input		The configuration of the auto topic route input, as contained in the tag <input> inside <auto_topic_route>
output		The configuration of the auto topic route output, as contained in the tag <output> inside <auto_topic_route>.
input.participant		Index of the input participant (1 or 2). The value of this field is used to determine whether the input of this auto topic route is the domain route's participant 1 or the participant 2.
input.allow_topic_name_ filter		Topics that do not pass this filter in the input participant will not trigger the creation of topic routes. This filter is configured using the tag <allow_topic_name_filter> inside <auto_topic_route>/<input>.
input.allow_registered_ type_name_filter		Topic with types that do not pass this filter in the input participant will not trigger the creation of topic routes. This filter is configured using the tag <allow_registered_type_name_filter> inside <auto_topic_route>/ <input>.
output.allow_topic_name_ filter		Topics that do not pass this filter in the output participant will not trigger the creation of topic routes. This filter is configured using the tag <allow_topic_name_filter> inside <auto_topic_route>/<output>.
output.allow_registered_ type_name_filter		Topics with types that do not pass this filter in the output participant will not trigger the creation of topic routes. This filter is configured using the tag <allow_registered_type_name_filter> inside <auto_topic_route>/ <output>.
input.deny_topic_name_ filter		Topics that pass this filter in the input participant will not trigger the creation of topic routes. This filter is configured using the tag <deny_topic_name_filter> inside <auto_topic_route>/<input>.

Table 5.5 **AutoRouteData**

Field Name	Key	Description
input.deny_registered_ type_name_filter		Topics with types that pass this filter in the input participant will not trigger the creation of topic routes. The input deny registered type name filter is configured using the tag <deny_registered_type_name_filter> inside <auto_topic_route>/<input>.
output.deny_topic_name_ filter		Topics that pass this filter in the output participant will not trigger the creation of topic routes. The output deny topic name filter is configured using the tag <deny_topic_name_filter> inside <auto_topic_route>/<output>.
output.deny_registered_ type_name_filter		Topics with types that pass this filter in the output participant will not trigger the creation of routes. The output deny registered type name filter is configured using the tag <deny_registered_type_name_filter> inside <auto_topic_route>/<output>.
input.creation_mode		Indicates when the DataReader is created in the input. The input creation mode is configured using the tag <creation_mode>.
output.creation_mode		Indicates when the DataWriter in created in the output. The output creation mode is configured using the tag <creation_mode>.
input.dds_data		Contains the configuration of the auto topic route's input that is specific to *Connext*.
output.dds_data		Contains the configuration of the auto_topic route's output that is specific to *Connext*.
input.dds_data.domain_id		Domain ID of the input participant
output.dds_data.domain_id		Domain ID of the output participant
input.dds_data. content_filter_expression		Content filter expression associated with the content filter for the topic route DataReader. The expression is configured using the tag <content_filter>/<expression> inside <topic_route>/<input>.

Table 5.5 **AutoRouteData**

Field Name	Key	Description
propagate_dispose		Indicates if the topic route propagates NOT_ALIVE_DISPOSE samples. The propagation of NOT_ALIVE_DISPOSE samples is configured using the tag <propagate_dispose> in <topic_route>.
propagate_unregister		Indicates if the topic routes propagate NOT_ALIVE_NO_WRITERS samples. The propagation of NOT_ALIVE_NO_WRITERS samples is configured using the tag <propagate_unregister> in <auto_topic_route>.
publish_with_original_info		Indicates if the topic routes publish the samples with original writer information. Setting this option to true allows redundant topic routes and prevents applications from receiving duplicate samples. The publication with original writer info is configured using the tag <publish_with_original_info> inside <auto_topic_route>.
publish_with_original_ timestamp		Indicates if the topic routes are configured to publish the output samples with the same timestamp as that of the input sample.
enabled_route_count		The number of enabled topic routes associated with the auto topic route.

5.3 Monitoring Status

Operational status for *Federation Service* entities is published in entity **status_set** topics. This information changes continuously and is computed and published periodically.

The status information for the different entities is composed primarily of statistics. Section 5.3.1 explains how these statistics are calculated and published. These sections describe the status information associated with each kind of entity:

- ❏ Status Information for the Federation Service (Section 5.3.2)
- ❏ Domain Route Status (Section 5.3.3)
- ❏ Status Information for a Session (Section 5.3.4)
- ❏ Status Information for a Topic Route (Section 5.3.5)
- ❏ Status Information for an Auto Route (Section 5.3.6)

Each section describes the IDL for the topics' underlying data types. The IDL is also in the file **<*Federation Service installation directory*>/resource/idl/FederationService-Monitoring.idl**.

5.3.1 How the Statistics are Generated

5.3.1.1 Statistics Publication

Federation Service reports multiple statistics as part of the different status sets. For example, for a topic route the status contains statistical metrics about the input and output samples per second (throughput).

```
struct RouteStatusSet {
    ...
    StatisticVariable input_samples_per_s;
    StatisticVariable output_samples_per_s;
    ...
};
```

The statistical information is published periodically in the form of **StatisticVariables**.

The period at which statistics are published is configurable using the tag <status_publication_ period> (see Section 2.4.4).

For a given variable, *Federation Service* computes the metrics in **StatisticMetrics** during specific time frames.

```
struct StatisticMetrics {
    unsigned long long period_ms;
    long long count;
    float mean;
    float minimum;
    float maximum;
    float std_dev;
};

struct StatisticVariable {
  StatisticMetric publication_period_metrics;
  sequence<StatisticMetrics, MAX_HISTORICAL_METRICS> historical_metrics;
};
```

The **count** is the sum of all the values received during the time frame. For example, in the case of **input_sample_per_s** and **output_sample_p_s**, **count** is the number of samples received during the time frame. For latency, **count** is the sum of all the latency times for the samples received during the time frame.

If status publication is enabled (see Section 2.4.4), *Federation Service* always publishes the statistics corresponding to the time between two status publications (**publication_period_metrics**). You can also select additional windows on a per entity basis using the tag <historical_statistics> (see Section 2.4.4). The sequence **historical_metrics** in **StatisticVariable** contains values corresponding to the windows that have been enabled:

- ❏ 5-sec. metrics correspond to activity in the last five seconds.
- ❏ 1-min. metrics correspond to activity in the last minute.
- ❏ 5-min. metrics correspond to activity in the last five minutes.
- ❏ 1-hour metrics correspond to activity in the last hour.
- ❏ Up-time metrics correspond to activity since the entity was enabled.

Each window has a field called **period_ms** that identifies its size in milliseconds. For the **publication_period_metrics,** this field contains the publication period. For the up-time metrics, this field contains the time since the entity was enabled. For the other windows, this field contains a fixed value that identifies the window size (5000 for the 5-second window, 60000 for the one-minute window, etc).

5.3.1.2 Statistics Calculation

The accuracy of the statistics calculation process is determined by the value of the statistics sampling period. This period specifies how often statistics are gathered and is configured on a per entity basis using the tag <statistics_sampling_ period> (see Section 2.4.4).

As a general rule, the **statistics_sampling_period** of an entity must be smaller than its **status_publication_period**. A small **statistics_sampling_period** provides more accurate statistics at expense of increasing the memory consumption and decreasing performance.

5.3.2 Status Information for the Federation Service

The topic that publishes federation service status is called **rti/federation_service/monitoring/federation_service_status _set**.

The IDL definition of the data type is:

```
struct FederationServiceStatusSet {
   string<ENTITY_NAME_MAX_LENGTH> name; //@key

   StatisticVariable cpu_usage_percentage;
```

```
    StatisticVariable physical_memory_kb;
    StatisticVariable total_memory_kb;
};
```

Table 5.6 describes the fields in the **FederationServiceStatusSet** data type.

Table 5.6 **FederationServiceStatusSet**

Field Name	Key	Description
name	✓	Name of the federation service instance. The name associated with the federation service instance can be assigned explicitly by using the **–appName** command-line parameter. If **–appName** is not used, the **<federation_service>** tag name provided with **-cfgName** is used. If you use the **–identifyExecution** command-line parameter, the host name and the process ID are appended to the name. For example: **RTI_FederationService_myhost_1234**
cpu_usage_percentage		Statistic variable that provides the percentage of CPU usage of the federation service process over different time windows. This variable is only supported on Windows and Linux systems.
physical_memory_kb		Statistic variable that provides the physical memory utilization of the federation service process. This variable is only supported on Windows and Linux systems.
total_memory_kb		Statistic variable that provides the virtual memory utilization of the federation service process. This variable is only supported on Windows and Linux systems.

5.3.3 Domain Route Status

The topic that publishes domain route status is called **rti/federation_service/monitoring/domain_route_status_set**.

The domain route status aggregates the statistics of the topic routes contained in it: the mean of the means in the topic routes, the absolute maximum and minimum across topic routes, the mean of the standard deviation and the total count.

The IDL definition of the data type is:

```
struct DomainRouteStatusSet {
    string<ENTITY_NAME_MAX_LENGTH> federation_service_name; //@key
    string<ENTITY_NAME_MAX_LENGTH> name; //@key

    StatisticVariable input_samples_per_s;
    StatisticVariable input_bytes_per_s;
    StatisticVariable output_samples_per_s;
    StatisticVariable output_bytes_per_s;
    StatisticVariable latency_s;
};
```

Table 5.7 describes the fields in the **DomainRouteStatusSet** data type.

Table 5.7 **DomainRouteStatusSet**

Field Name	Key	Description
federation_service_name	✓	The federation service name (assigned with **–appName**).
name	✓	The domain route name. The domain route name is configured using the name attribute in the **<domain_route>** tag.
input_samples_per_s		Statistic variable that provides information about the input samples per second across topic routes. Input samples refer to the samples that are taken by the sessions from the topic routes's DataReaders.
input_bytes_per_s[1]		Statistic variable that provides information about the input bytes per second across topic routes. Input bytes refer to the bytes that are taken by the sessions from the topic routes's DataReaders. These bytes only refer to the serialized samples. The protocol headers (UDP, RTPS) are not included.
output_samples_per_s		Statistic variable that provides information about the output samples per second across topic routes. Output samples refer to the samples that are published out by the session threads using the topic route's DataWriters.

Table 5.7 **DomainRouteStatusSet**

Field Name	Key	Description
output_bytes_per_s		Statistic variable that provides information about the output bytes per second across topic routes. Output bytes refer to the bytes that are published out by the session threads using the topic route's DataWriters. The variable only considers the bytes of the serialized samples. Protocol headers (UDP, RTPS) are not included.
latency_s		Statistic variable that provides information about the latency in seconds across topic routes. The latency in a topic route refers to the time elapsed between the sample read and write.

1. The throughput measured in bytes can only be computed if the samples are DynamicData samples. If not, only the throughput measured in samples per second is available. This statement applies to all the statistic variables described in this chapter that measure throughput in bytes per second.

5.3.4 Status Information for a Session

The topic that publishes session status is called **rti/federation_service/monitoring/session_status_set**.

The session status aggregates the statistics of the topic routes contained in it: the mean of the means in the topic routes, the absolute maximum and minimum across topic routes, the mean of the standard deviation and the total count.

The IDL definition of the data type is:

```
struct SessionStatusSet {
    string<ENTITY_NAME_MAX_LENGTH> federation_service_name; //@key
   string<ENTITY_NAME_MAX_LENGTH> domain_route_name; //@key
   string<ENTITY_NAME_MAX_LENGTH> name; //@key

   StatisticVariable input_samples_per_s;
   StatisticVariable input_bytes_per_s;
   StatisticVariable output_samples_per_s;
   StatisticVariable output_bytes_per_s;
   StatisticVariable latency_s;
};
```

Table 5.8 describes the fields in the **SessionStatusSet** data type.

Table 5.8 **SessionStatusSet**

Field Name	Key	Description
federation_service_name	✓	The federation service name (assigned with **–appName**).
domain_route_name	✓	The domain route name
name	✓	The session name. The domain route name is configured using the name attribute in the **<session>** tag.
input_samples_per_s		Statistic variable that provides information about the input samples per second across topic routes. Input samples refer to the samples that are taken by the session from the topic routes's DataReaders.
input_bytes_per_s		Statistic variable that provides information about the input bytes per second across topic routes. Input bytes refer to the bytes that are taken by the sessions from the topic routes's DataReaders. These bytes only refer to the serialized samples. The protocol headers (UDP, RTPS) are not included.
output_samples_per_s		Statistic variable that provides information about the output samples per second across topic routes. Output samples refer to the samples that are published out by the session thread using the topic route's DataWriters.
output_bytes_per_s		Statistic variable that provides information about the output bytes per second across topic routes. Output bytes refer to the bytes that are published out by the session thread using the topic route's DataWriters. The variable only considers the bytes of the serialized samples. Protocol headers (UDP, RTPS) are not included.
latency_s		Statistic variable that provides information about the latency in seconds across topic routes. The latency in a topic route refers to the time elapsed between the sample read and write.

5.3.5 Status Information for a Topic Route

The topic that publishes route status is called **rti/federation_service/monitoring/route_status_set**.

The IDL definition of the data type is:

```
struct RouteStatusSet {
    string<ENTITY_NAME_MAX_LENGTH> federation_service_name; //@key
   string<ENTITY_NAME_MAX_LENGTH> domain_route_name; //@key
   string<ENTITY_NAME_MAX_LENGTH> session_name; //@key
   string<ENTITY_NAME_MAX_LENGTH> name; //@key

   StatisticVariable input_samples_per_s;
   StatisticVariable input_bytes_per_s;
   StatisticVariable output_samples_per_s;
   StatisticVariable output_bytes_per_s;
   StatisticVariable latency_s;
};
```

Table 5.9 describes the fields in the **RouteStatusSet** data type.

Table 5.9 **RouteStatusSet**

Member Name	Key	Description
federation_service_name	✓	The federation service name (assigned with **-appName**).
domain_route_name	✓	The domain route name
session_name	✓	The session name.
name	✓	The topic route name. The topic route name is configured using the name attribute in the <topic_route> tag.
input_samples_per_s		Statistic variable that provides information about the input samples per second in the topic route. Input samples refer to the samples that are taken by the session from the topic route's DataReader.
input_bytes_per_s		Statistic variable that provides information about the input bytes per second in the topic route. Input bytes refer to the bytes that are taken by the session from the topic route's DataReader. These bytes only refer to the serialized samples. The protocol headers (UDP, RTPS) are not included.
output_samples_per_s		Statistic variable that provides information about the output samples per second in the topic route. Output samples refer to the samples that are published out by the session thread using the topic route's DataWriter.

Table 5.9 **RouteStatusSet**

Member Name	Key	Description
output_bytes_per_s		Statistic variable that provides information about the output bytes per second in the topic route. Output bytes refer to the bytes that are published out by the session thread using the topic route's DataWriter. The variable only considers the bytes of the serialized samples. Protocol headers (UDP, RTPS) are not included.
latency_s		Statistic variable that provides information about the latency in seconds in the topic route. The latency in a topic route refers to the time elapsed between the sample read and write.

5.3.6 Status Information for an Auto Route

The topic that publishes auto topic route status is called **rti/federation_service/monitoring/auto_route_status_set**.

The auto route status aggregates the statistics of the routes created from it: the mean of the means in the routes, the absolute maximum and minimum across routes, the mean of the standard deviation and the total count.

The IDL definition of the data type is:

```
struct AutoRouteStatusSet {
    string<ENTITY_NAME_MAX_LENGTH> federation_service_name; //@key
   string<ENTITY_NAME_MAX_LENGTH> domain_route_name; //@key
   string<ENTITY_NAME_MAX_LENGTH> session_name; //@key
   string<ENTITY_NAME_MAX_LENGTH> name; //@key

   StatisticVariable input_samples_per_s;
   StatisticVariable input_bytes_per_s;
   StatisticVariable output_samples_per_s;
   StatisticVariable output_bytes_per_s;
   StatisticVariable latency_s;
};
```

Table 5.10 describes the fields in the **AutoRouteStatusSet** data type.

Table 5.10 **AutoRouteStatusSet**

Member Name	Key	Description
federation_service_name	✓	The federation service name (assigned with **–appname**).
domain_route_name	✓	The domain route name.
session_name	✓	The session name.
name	✓	The auto topic route name. The auto topic route name is configured using the name attribute in the **<auto_topic_route>** tag.
input_samples_per_s		Statistic variable that provides information about the input samples per second across topic routes. Input samples refer to the samples that are taken by the session from the auto topic routes's DataReaders.
input_bytes_per_s		Statistic variable that provides information about the input bytes per second across topic routes. Input bytes refer to the bytes that are taken by the session from the auto topic routes's DataReaders. These bytes only refer to the serialized samples. The protocol headers (UDP, RTPS) are not included.
output_samples_per_s		Statistic variable that provides information about the output samples per second across topic routes. Output samples refer to the samples that are published out by the session thread using the auto topic route's DataWriters.
output_bytes_per_s		Statistic variable that provides information about the output bytes per second across topic routes. Output bytes refer to the bytes that are published out by the session thread using the auto topic route's DataWriters. The variable only considers the bytes of the serialized samples. Protocol headers (UDP, RTPS) are not included.
latency_s		Statistic variable that provides information about the latency in seconds across topic routes. The latency in a route refers to the time elapsed between the sample read and write.

Chapter 6 Traversing Wide Area Networks

Many systems today already rely on *Connext* to distribute their information across a Local Area Network (LAN). However, more and more of these systems are being integrated in Wide Area Networks (WANs). With *Federation Service*, you can scale Connext applications beyond the current local networks and make it available throughout a WAN.

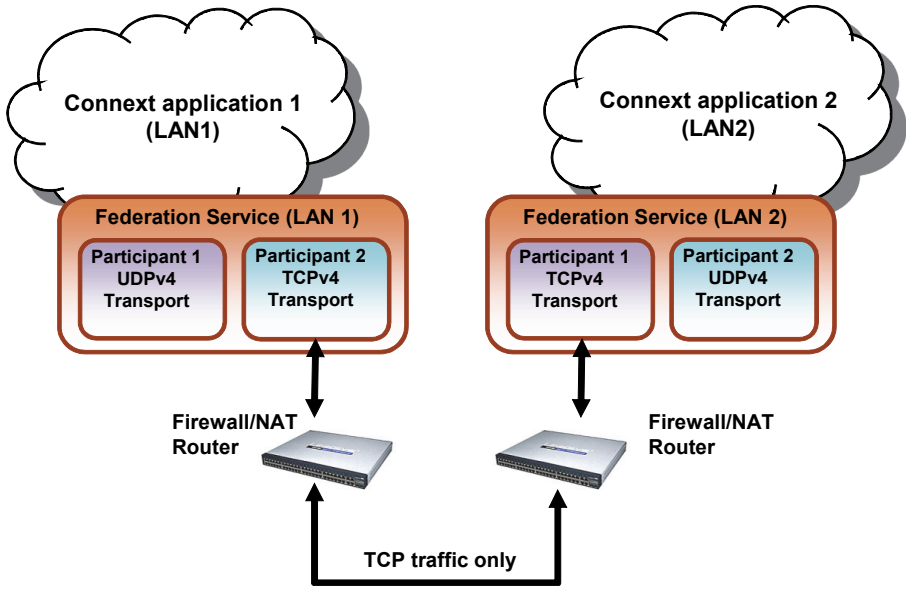
Out of the box, *Federation Service* only uses UDPv4 and Shared Memory transports to communicate with other *Federation Service*s and *Connext* applications. This configuration is appropriate for systems running within a single LAN. However, the usage of UDPv4 introduces several problems when we try to communicate Connext applications running in different LANs:

- ❏ UDPv4 traffic is usually filtered out by the LAN firewalls for security reasons.
- ❏ Forwarded ports are usually TCP ports.
- ❏ Each LAN may run in its own private IP address space and use NAT (Network Address Translation) to communicate with other networks.

To overcome these three issues, *Federation Service* is distributed with a TCP transport that is NAT friendly. The transport can be configured via XML using the PropertyQosPolicy of the *Federation Service's* participants. Figure 6.1 shows a typical scenario where two *Federation Services* are used to bridge two *Connext* applications running in two different LANs.

In the next sections, we will explain how to use and configure the TCP transport with *Federation Service*.

Figure 6.1 **WAN Communication Using TCP Transport**



6.1 TCP Communication Scenarios

The TCP transport distributed with *Federation Service* can be used to address multiple communication scenarios that go from simple communication within a single LAN to complex communication scenarios across LANs where NATs and firewalls may be involved.

6.1.1 Communication Within a Single LAN

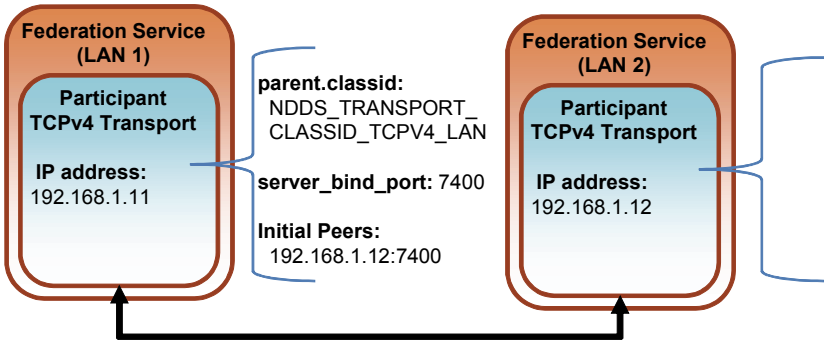
TCP transport can be used as an alternative to UDPv4 to communicate *Connext* applications running inside the same LAN.

Figure 6.2 shows how to configure the TCP transport in this scenario.

parent.classid, **transport_mode** and **server_bind_port** are transport properties configured using the PropertyQosPolicy of the participant.

Initial Peers represents the peers to which the participant will be announced to. Usually, these peers are configured using the DiscoveryQosPolicy of the participant or the environment variable NDDS_DISCOVERY_PEERS. For information on the format of initial peers, see Section 6.2.1.

Figure 6.2 **Communication within a Single LAN**



6.1.2 Symmetric Communication Across NATs

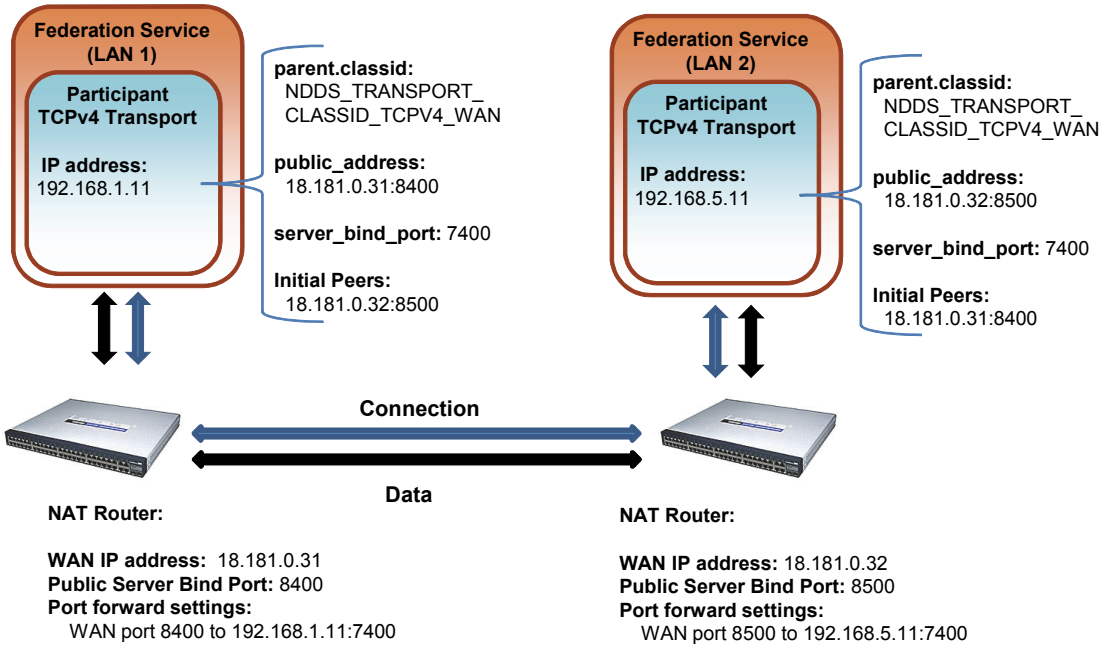
In NAT communication scenarios, each one of the LANs has a private IP address space. The communication with other LANs is done through NAT routers that translate private IP addresses and ports into public IP addresses and ports.

In symmetric communication scenarios, any instance of *Federation Service* can initiate TCP connections with other federation services. Figure 6.3 shows how to configure the TCP transport in this scenario.

Notice that initial peers refer to the public address of the *Federation Service* instances and not the LAN address. In addition, the transport associated with a *Federation Service* instance will have to be configured with its **public_address** so that this information can be propagated as part of the discovery process.

Because the public address and port of the *Federation Service* instances must be known before the communication is established, the NAT Routers will have to be configured statically to translate (forward) the private **server_bind_port** into a public **port**. This process is known as “static NAT“ or “port forwarding” and it allows traffic originating in outer networks to reach designated peers in the LAN behind the NAT router.

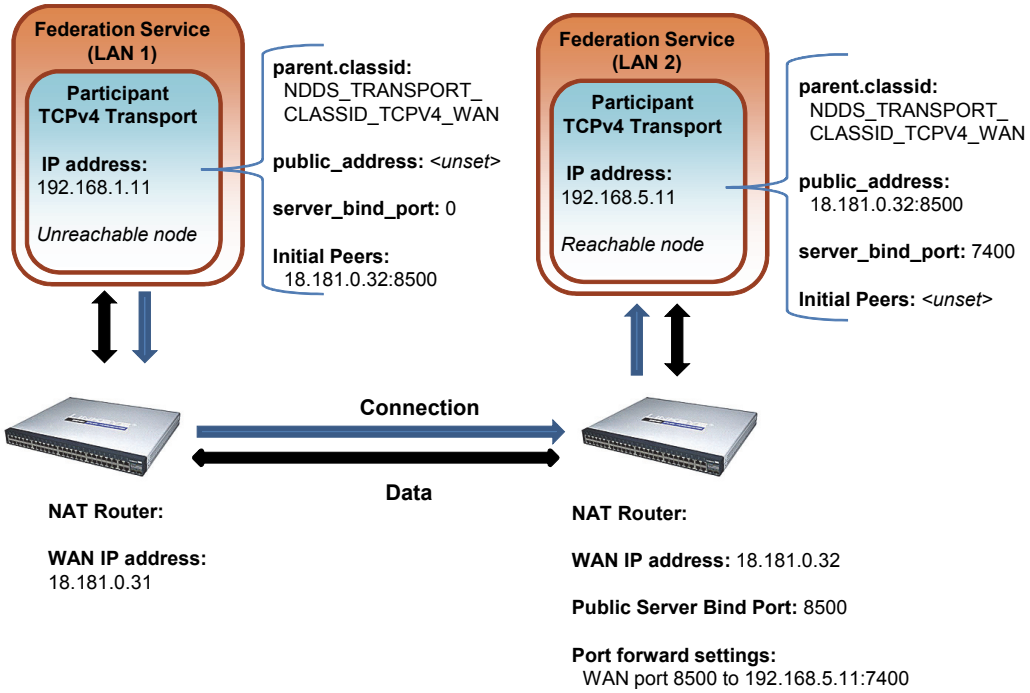
Figure 6.3 **Symmetric Communication across NATs**



6.1.3 Asymmetric Communication Across NATs

This scenario is similar to the previous one, except in this case the TCP connections can be initiated only by the *Federation Service* instance in LAN1. For security reasons, incoming connections to LAN1 are not allowed. Figure 6.4 shows how to configure the TCP transport in this scenario.

Figure 6.4 **Asymmetric Communication Across NATs**



Notice that the Federation Service on LAN 1 now does not have a public_address set (and its server_bind_port is set to zero), meaning that it cannot be reached from the outside network.

6.1.4 Secure Communication

Security can be added on top of any of the above scenarios. You can have secure communication within the same LAN or across NATs.

To enable secure communication, modify the previous configurations as follows:

- ❏ Change the transport class ID property (**parent.classid**) to be one of the following values:
  NDDS_TRANSPORT_CLASSID_TLSV4_LAN
  NDDS_TRANSPORT_CLASSID_TLSV4_WAN
- ❏ Set at least a certificate of authority (through either the **tls.verify.ca_file** or **tls.verify.ca_path** properties), and the certificate identity (through either the **tls.identity.certificate_chain**, or **tls.identity.certificate_chain_file** properties)
- ❏ Make sure to use '**tlsv4_lan**' or '**tlsv4_wan**' in the initial peers list as the prefix for all destination addresses.

To see the differences between a WAN scenario and the same scenario with TLS enabled, you can compare the two example configuration files:

- ❏ **shapes/tcp_transport.xml**
- ❏ **shapes/tcp_transport_tls.xml**

6.2 Configuring the TCP Transport

The TCP transport is distributed as a shared library in ***<Federation Service installation directory>*/bin/<*architecture*>.** The library is called **nddstransporttcp.dll** on Windows and **libnddstransporttcp.so** on UNIX-based systems.

For an example on how to use and configure the TCP transport with *Federation Service* see Example 5 - Using TCP Transport with Federation Service (Section 4.5) in the Getting Started Guide.

As seen in the example, you can configure the properties of the transport in the XML configuration file using the appropriate name/value pairs in the DomainParticipant's PropertyQoSPolicy. This will cause *Federation Service* to dynamically load the TCP transport library at run time and then implicitly create and register the transport plugin with *Connext*.

6.2.1 TCP Transport Initial Peers

With the TCP transport, the addresses of the initial peers (NDDS_DISCOVERY_PEERS) that will be contacted during the discovery process have the following format:

```
For WAN communication: tcpv4_wan://<IP address or hostname>:<port>
For LAN communication: tcpv4_lan://<IP address or hostname>:<port>
For WAN+TLS communication: tlsv4_wan://<IP address or hostname>:port
For LAN+TLS communication: tlsv4_lan://<IP address or hostname>:port
```

For example:

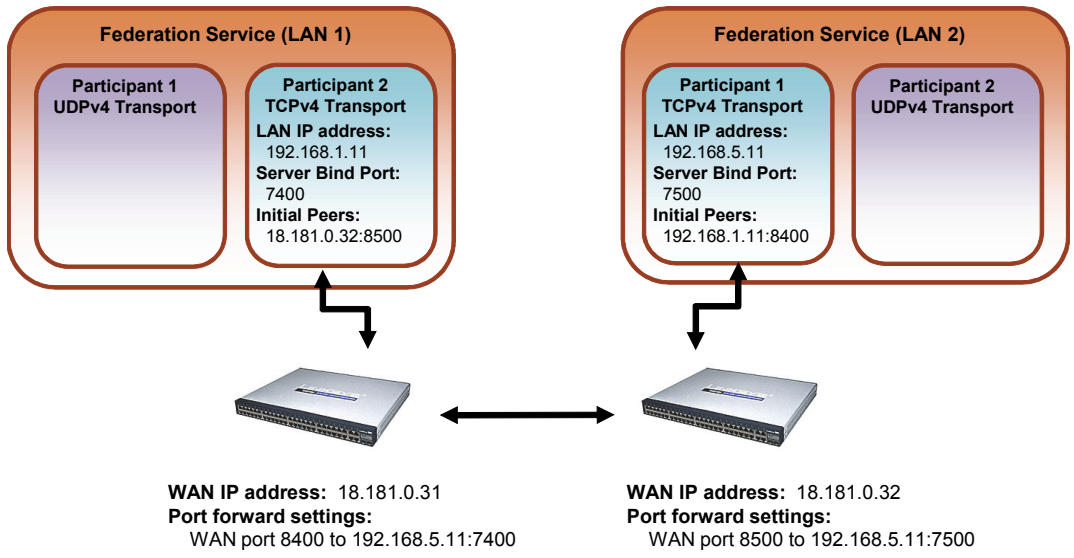
```
setenv NDDS_DISCOVERY_PEERS tcpv4_wan://10.10.1.165:7400,tcpv4_wan:/
/10.10.1.111:7400,tcpv4_lan://192.168.1.1:7500
```

When the TCP transport is configured for LAN communication (with the **parent.classid** property), the IP address is the LAN address of the peer and the port is the server port used by the transport (the **server_bind_port** property).

When the TCP transport is configured for WAN communication (with the **parent.classid** property), the IP address is the WAN or public address of the peer and the port is the public port that is used to forward traffic to the server port in the TCP transport.

When TLS is enabled, the transport settings are similar to WAN and LAN over TCP.

Figure 6.5 **Initial Peers in WAN Communication**



6.2.2 Setting Up the TCP Transport Properties with the PropertyQoSPolicy

The PropertyQosPolicy allows you to set up name/value pairs of data and attach them to an entity, such as a DomainParticipant. The configuration of the TCP transport with *Federation Service* is done using the PropertyQosPolicy of the Domain Participants that are going to use the transport.

For a list of the properties that you can set for the TCP transport, see the *RTI Core Libraries and Utilities User's Manual.*

In the following example, participant_1 will communicate with other participants on the same LAN using UDP and Shared Memory transports; participant_2 will communicate with other participants in different LANs using the TCP transport.

```
<dds>
    <federation_service name="MyFederationService">
        <domain_route name="MyDomainRoute">
            <participant_1>
                <domain_id>56</domain_id>
            </participant_1>
            <participant_2>
                <domain_id>57</domain_id>
                <participant_qos>
                    <transport_builtin>
                        <mask>MASK_NONE</mask>
                    </transport_builtin>
                    <property>
                    <value>
                        <element>
                            <name>dds.transport.load_plugins</name>
                            <value>dds.transport.TCPv4.tcp1</value>
                        </element>
                        <element>
                            <name>
                                dds.transport.TCPv4.tcp1.library
                            </name>
                            <value>libnddstransporttcp.so</value>
                        </element>
                        <element>
                            <name>
                                dds.transport.TCPv4.tcp1.create_function
                            </name>
                            <value>NDDS_Transport_TCPv4_create</value>
                        </element>
```

```
                        <element>
                            <name>
                                dds.transport.TCPv4.tcp1.parent.classid
                            </name>
                            <value>
                                NDDS_TRANSPORT_CLASSID_TCPV4_WAN
                            </value>
                        </element>
                        <element>
                            <name>
                                dds.transport.TCPv4.tcp1.public_address
                            </name>
                            <value>18.181.0.31:8400</value>
                        </element>
                        <element>
                            <name>
                                dds.transport.TCPv4.tcp1.server_bind_port
                            </name>
                            <value>7400</value>
                        </element>
                    </value>
                </property>
            </participant_qos>
            </participant_2>
        </domain_route>
    </federation_service>
</dds>
```

6.2.3 TCP/TLS Transport Properties

Table 6.1 describes the TCP and TLS transport properties.

Table 6.1 **TCP/TLS Transport Properties (over LAN or WAN) — NDDS_Transport_TCPv4_Property_t**

Property Name (prefix with 'dds.transport.TCPv4.tcp1.')[1]	Description
dds.transport.load_plugins **(Note: this does not take a prefix)**	**Required** Comma-separated strings indicating the prefix names of all plugins that will be loaded by *Connext*. For example: "**dds.transport.TCPv4.tcp1**". You will use this string as the prefix to the property names. See Footnote 1 on page 6-20. Note: you can load up to 8 plugins.

Table 6.1 **TCP/TLS Transport Properties (over LAN or WAN) — NDDS_Transport_TCPv4_Property_t**

Property Name (prefix with 'dds.transport.TCPv4.tcp1.')[1]	Description
library	**Required** **Must be "nddstransporttcp".** This library needs to be in the path during run time (in the LD_LIBRARY_PATH environment variable on UNIX systems, in PATH for Windows systems).
create_function	**Required** **Must be "NDDS_Transport_TCPv4_create".**
aliases	Used to register the transport plugin returned by **NDDS_Transport_TCPv4_create()** (as specified by **<TCP_prefix>.create_function**) to the *DomainParticipant*. Aliases should be specified as a comma-separated string, with each comma delimiting an alias. Default: the transport prefix (see Footnote [1])
parent.classid	Must be set to one of the following values: **NDDS_TRANSPORT_CLASSID_TCPV4_LAN** for TCP communication within a LAN **NDDS_TRANSPORT_CLASSID_TLSV4_LAN** for TLS communication within a LAN **NDDS_TRANSPORT_CLASSID_TCPV4_WAN** for TCP communication across LANs and firewalls **NDDS_TRANSPORT_CLASSID_TLSV4_WAN** for TLS communication across LAN and firewalls Default: **NDDS_TRANSPORT_CLASSID_TCPV4_LAN**
parent.gather_send_buffer_count_max	Specifies the maximum number of buffers that *Connext* can pass to the **send()** function of the transport plugin. The transport plugin **send()** operation supports a gather-send concept, where the **send()** call can take several discontiguous buffers, assemble and send them in a single message. This enables *Connext* to send a message from parts obtained from different sources without first having to copy the parts into a single contiguous buffer. However, most transports that support a gather-send concept have an upper limit on the number of buffers that can be gathered and sent. Setting this value will prevent *Connext* from trying to gather too many buffers into a send call for the transport plugin. *Connext* requires all transport-plugin implementations to support a gather-send of least a minimum number of buffers. This minimum number is defined as NDDS_TRANSPORT_PROPERTY_GATHER_SEND_BUFFER_COUNT_MIN Default: 128

Table 6.1 **TCP/TLS Transport Properties (over LAN or WAN) — NDDS_Transport_TCPv4_Property_t**

Property Name (prefix with 'dds.transport.TCPv4.tcp1.')[1]	Description
parent. message_size_max	The maximum size of a message, in bytes, that can be sent or received by the transport plugin. If you set this higher than the default, the *DomainParticipant's* **buffer_size** (in the RECEIVER_POOL QosPolicy, see the *RTI Core Libraries and Utilities User's Manual*) should also be changed. Default: 9216
parent. allow_interfaces_list	A list of strings, each identifying a range of interface addresses that can be used by the transport. Interfaces must be specified as comma-separated strings, with each comma delimiting an interface. For example: 10.10.*, 10.15.* If the list is non-empty, this "white" list is applied before **parent. deny_interfaces_list**. Default: All available interfaces are used.
parent. deny_interfaces_list	A list of strings, each identifying a range of interface addresses that will not be used by the transport. If the list is non-empty, deny the use of these interfaces. Interfaces must be specified as comma-separated strings, with each comma delimiting an interface. For example: `10.10.*` This "black" list is applied after **parent. allow_interfaces_list** and filters out the interfaces that should not be used. **Default:** No interfaces are denied
send_socket_buffer_size	Size, in bytes, of the send buffer of a socket used for sending. On most operating systems, **setsockopt()** will be called to set the SENDBUF to the value of this parameter. This value must be greater than or equal to **parent. message_size_max** or -1. The maximum value is operating system-dependent. Default: -1 (**setsockopt()** (or equivalent) will not be called to size the send buffer of the socket)

Table 6.1 **TCP/TLS Transport Properties (over LAN or WAN) — NDDS_Transport_TCPv4_Property_t**

Property Name (prefix with 'dds.transport.TCPv4.tcp1.')[1]	Description
recv_socket_buffer_size	Size, in bytes, of the receive buffer of a socket used for receiving. On most operating systems, **setsockopt()** will be called to set the RECVBUF to the value of this parameter. This value must be greater than or equal to **parent. message_size_max** or -1. The maximum value is operating-system dependent. Default: -1 (**setsockopt()** (or equivalent) will not be called to size the receive buffer of the socket)
ignore_loopback_interface	Prevents the transport plugin from using the IP loopback interface. This property is ignored when **parent.classid** is NDDS_TRANSPORT_CLASSID_TCPV4_WAN or NDDS_TRANSPORT_CLASSID_TLSV4_WAN. Two values are allowed: 0: Enable local traffic via this plugin. The plugin will use and report the IP loopback interface only if there are no other network interfaces (NICs) up on the system. 1: Disable local traffic via this plugin. This means "do not use the IP loopback interface, even if no NICs are discovered." This setting is useful when you want applications running on the same node to use a more efficient plugin like shared memory instead of the IP loopback. Default: 1
ignore_nonrunning_interfaces	Prevents the transport plugin from using a network interface that is not reported as RUNNING by the operating system. The transport checks the flags reported by the operating system for each network interface upon initialization. An interface which is not reported as UP will not be used. This property allows the same check to be extended to the IFF_RUNNING flag implemented by some operating systems. The RUNNING flag means that "all resources are allocated" and may be off if no link is detected (e.g., the network cable is unplugged). Two values are allowed: 0: Do not check the RUNNING flag when enumerating interfaces, just make sure the interface is UP. 1: Check the flag when enumerating interfaces, and ignore those that are not reported as RUNNING. This can be used on some operating systems to cause the transport to ignore interfaces that are enabled but not connected to the network. Default: 1

Table 6.1 **TCP/TLS Transport Properties (over LAN or WAN) — NDDS_Transport_TCPv4_Property_t**

Property Name (prefix with 'dds.transport.TCPv4.tcp1.')[1]	Description
transport_priority_mask	Mask for the transport priority field. This is used in conjunction with **transport_priority_ mapping_low**/**transport_priority_ mapping_high** to define the mapping from *Connext* transport priority to the IPv4 TOS field. Defines a contiguous region of bits in the 32-bit transport priority value that is used to generate values for the IPv4 TOS field on an outgoing socket. For example, the value 0x0000ff00 causes bits 9-16 (8 bits) to be used in the mapping. The value will be scaled from the mask range (0x0000 -0xff00 in this case) to the range specified by low and high. If the mask is set to zero, then the transport will not set IPv4 TOS for send sockets. Default: 0
transport_priority_ mapping_low	Sets the low and high values of the output range to IPv4 TOS. These values are used in conjunction with **transport_priority_mask** to define the mapping from *Connext* transport priority to the IPv4 TOS field. Defines the low and high values of the output range for scaling. Note that IPv4 TOS is generally an 8-bit value. Default transport_priority_mapping_low: 0 Default transport_priority_mapping_high: 0xFF
transport_priority_ mapping_high	
server_socket_backlog	Determines the maximum length of the queue of pending connections. Default: 5
public_address	**Required for WAN communication** Public IP address and port (WAN address and port) associated with the transport instantiation.The address and port must be separated with ':'. For example: 10.10.9.10:4567 This field is only used when **parent.classid** is NDDS_TRANSPORT_CLASSID_TCPV4_WAN or NDDS_TRANSPORT_CLASSID_TLSV4_WAN. The public address and port are necessary to support communication over a WAN that involves Network Address Translators (NATs). Typically, the address is the public address of the IP router that provides access to the WAN. The port is the IP router port that is used to reach the private **server_bind_port** inside the LAN from the outside. This value is expressed as a string in the form: **ip[:port],** where **ip** represents the IPv4 address and **port** is the external port number of the router. Note that host names are not allowed in the **public_address** because they may resolve to an internet address that is not what you want (i.e., 'localhost' may map to your local IP or to 127.0.0.1).

Table 6.1 **TCP/TLS Transport Properties (over LAN or WAN) — NDDS_Transport_TCPv4_Property_t**

Property Name (prefix with 'dds.transport.TCPv4.tcp1.')[1]	Description
server_bind_port	Private IP port (inside the LAN) used by the transport to accept TCP connections. If this property is set to zero, the transport will disable the internal server socket, making it impossible for external peers to connect to this node. In this case, the node is considered unreachable and will communicate only using the asynchronous mode with other (reachable) peers. For WAN communication, this port must be forwarded to a public port in the NAT-enabled router that connects to the outer network. Default: 7400
read_buffer_allocation	Allocation settings applied to read buffers. These settings configure the initial number of buffers, the maximum number of buffers and the buffers to be allocated when more buffers are needed. Default: ❑ **read_buffer_allocation.initial_count** = 2 ❑ **read_buffer_allocation.max_count** = -1 (unlimited) ❑ **read_buffer_allocation.incremental_count** = -1 (number of buffers will keep doubling on each allocation until it reaches **max_count**)
write_buffer_allocation	Allocation settings applied to buffers used for an asynchronous (non-blocking) write. These settings configure the initial number of buffers, the maximum number of buffers, and the buffers to be allocated when more buffers are needed. Default: ❑ **write_buffer_allocation.initial_count** = 4 ❑ **write_buffer_allocation.max_count** = 1000 ❑ **write_buffer_allocation.incremental_count** = 10 Note that for the write buffer pool, the **max_count** is not set to unlimited. This is to avoid having a fast writer quickly exhaust all the available system memory, in case of a temporary network slowdown. When this write buffer pool reaches the maximum, the low-level send command of the transport will fail; at that point *Connext* will take the appropriate action (retry to send or drop it), according to the application's QoS (if the transport is used for reliable communication, the data will still be sent eventually).

Table 6.1 **TCP/TLS Transport Properties (over LAN or WAN) — NDDS_Transport_TCPv4_Property_t**

Property Name (prefix with 'dds.transport.TCPv4.tcp1.')[1]	Description
control_buffer_allocation	Allocation settings applied to buffers used to serialize and send control messages. These settings configure the initial number of buffers, the maximum number of buffers, and the buffers to be allocated when more buffers are needed. Default: ❑ **control_buffer_allocation.initial_count** = 2 ❑ **control_buffer_allocation.max_count** = -1 (unlimited) ❑ **control_buffer_allocation.incremental_count** = -1 (number of buffers will keep doubling on each allocation until it reaches **max_count**)
control_message_allocation	Allocation settings applied to control messages. These settings configure the initial number of messages, the maximum number of messages, and the messages to be allocated when more messages are needed. Default: ❑ **control_message_allocation.initial_count** = 2 ❑ **control_message_allocation.max_count** = -1 (unlimited) ❑ **control_message_allocation.incremental_count** = -1 (number of messages will keep doubling on each allocation until it reaches **max_count**)
control_attribute_allocation	Allocation settings applied to control messages attributes. These settings configure the initial number of attributes, the maximum number of attributes, and the attributes to be allocated when more attributes are needed. Default: ❑ **control_attribute_allocation.initial_count** = 2 ❑ **control_attribute_allocation.max_count** = -1 (unlimited) ❑ **control_attribute_allocation.incremental_count** = -1 (number of attributes will keep doubling on each allocation until it reaches **max_count**)

Table 6.1 **TCP/TLS Transport Properties (over LAN or WAN) — NDDS_Transport_TCPv4_Property_t**

Property Name (prefix with 'dds.transport.TCPv4.tcp1.')[1]	Description
force_asynchronous_send	Forces an asynchronous send. When this parameter is set to 0, the TCP transport will attempt to send data as soon as the internal **send()** function is called. When it is set to 1, the transport will make a copy of the data to send and enqueue it in an internal send buffer. Data will be sent as soon as the low-level socket buffer has space. Normally setting it to 1 delivers better throughput in a fast network, but will result in a longer time to recover from various TCP error conditions. Setting it to 0 may cause the low-level **send()** function to block until the data is physically delivered to the lower socket buffer. For an application writing data at a very fast rate, it may cause the caller thread to block if the send socket buffer is full. This could produce lower throughput in those conditions (the caller thread could prepare the next packet while waiting for the send socket buffer to become available). Default: 0
max_packet_size	The maximum size of a TCP segment. This parameter is only supported on Linux architectures. By default, the maximum size of a TCP segment is based on the network MTU for destinations on a local network, or on a default 576 for destinations on non-local networks. This behavior can be changed by setting this parameter to a value between 1 and 65535. Default: -1 (default behavior)
enable_keep_alive	Configures the sending of KEEP_ALIVE messages in TCP. Setting this value to 1 causes a KEEP_ALIVE packet to be sent to the remote peer if a long time passes with no other data sent or received. This feature is implemented only on architectures that provide a low-level implementation of the TCP keep-alive feature. On Windows systems, the TCP keep-alive feature can be globally enabled through the system's registry: **\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Tcpip\Parameters**. Refer to MSDN documentation for more details. On Solaris systems, most of the TCP keep-alive parameters can be changed though the kernel properties. Default: 0

Table 6.1 **TCP/TLS Transport Properties (over LAN or WAN) — NDDS_Transport_TCPv4_Property_t**

Property Name (prefix with 'dds.transport.TCPv4.tcp1.')[1]	Description
keep_alive_time	Specifies the interval of inactivity, in seconds, that causes TCP to generate a KEEP_ALIVE message. This parameter is only supported on Linux architectures. Default: -1 (OS default value)
keep_alive_interval	Specifies the interval, in seconds, between KEEP_ALIVE retries. This parameter is only supported on Linux architectures. Default: -1 (OS default value)
keep_alive_retry_count	The maximum number of KEEP_ALIVE retries before dropping the connection. This parameter is only supported on Linux architectures. Default: -1 (OS default value)
disable_nagle	Disables the TCP nagle algorithm. When this property is set to 1, TCP segments are always sent as soon as possible, which may result in poor network utilization. Default: 0

Table 6.1 **TCP/TLS Transport Properties (over LAN or WAN) — NDDS_Transport_TCPv4_Property_t**

Property Name (prefix with 'dds.transport.TCPv4.tcp1.')[1]	Description
logging_verbosity_bitmap	Bitmap that specifies the verbosity of log messages from the transport. Logging values: ❏ -1 (0xffffffff): do not change the current verbosity ❏ 0x00: silence ❏ 0x01: errors ❏ 0x02: warnings ❏ 0x04: local ❏ 0x08: remote ❏ 0x10: period ❏ 0x80: other (used for control protocol tracing) Default: -1 Note: the logging verbosity is a global property shared across multiple instances of the TCP transport. If you create a new TCP Transport instance with **logging_verbosity_bitmap** different than -1, the change will affect all the other instances as well. The default TCP transport verbosity is errors and warnings. Note: The option of 0x80 (other) is used only for tracing the internal control protocol. Since the output is very verbose, this feature is enabled only in the debug version of the *TCP Transport* library (**libnddstransporttcpd.so / LIBNDDSTRANSPORTD.LIB**).
outstanding_connection_cookies	Maximum number of outstanding connection cookies allowed by the transport when acting as server. A connection cookie is a token provided by a server to a client; it is used to establish a data connection. Until the data connection is established, the cookie cannot be reused by the server. To avoid wasting memory, it is good practice to set a cap on the maximum number of connection cookies (pending connections). When the maximum value is reached, a client will not be able to connect to the server until new cookies become available. Range: 1 or higher, or -1 (which means an unlimited number). Default: 100

Table 6.1 **TCP/TLS Transport Properties (over LAN or WAN) — NDDS_Transport_TCPv4_Property_t**

Property Name (prefix with 'dds.transport.TCPv4.tcp1.')[1]	Description
outstanding_connection_cookies_life_span	Maximum lifespan (in seconds) of the cookies associated with pending connections. If a client does not connect to the server before the lifespan of its cookie expires, it will have to request a new cookie. Range: 1 second or higher, or -1 Default : -1, which means an unlimited amount of time (effectively disabling the feature).
tls.verify.ca_file	A string that specifies the name of a file containing Certificate Authority certificates. The file should be in PEM format. See the OpenSSL manual page for SSL_load_verify_locations for more information. To enable TLS, ca_file or ca_path is required; both may be specified (at least one is required).
tls.verify.ca_path	A string that specifies paths to directories containing Certificate Authority certificates. Files should be in PEM format and follow the OpenSSL-required naming conventions. See the OpenSSL manual page for SSL_CTX_load_verify_locations for more information. To enable TLS, ca_file or ca_path is required; both may be specified (at least one is required).
tls.verify.verify_depth	Maximum certificate chain length for verification.
tls.verify.crl_file	Name of the file containing the Certificate Revocation List. File should be in PEM format.
tls.cipher.cipher_list	List of available TLS ciphers. See the OpenSSL manual page for SSL_set_cipher_list for more information on the format of this string.
tls.cipher.dh_param_files	List of available Diffie-Hellman (DH) key files. For example: "foo.pem:512,bar.pem:256" means: dh_param_files[0].file = foo.pem, dh_param_files[0].bits = 512, dh_param_files[1].file = bar.pem, dh_param_files[1].bits = 256
tls.cipher.engine_id	String ID of OpenSSL cipher engine to request.

Table 6.1 **TCP/TLS Transport Properties (over LAN or WAN) — NDDS_Transport_TCPv4_Property_t**

Property Name (prefix with 'dds.transport.TCPv4.tcp1.')[1]	Description
tls.identity.certificate_chain	A string containing an identifying certificate chain (in PEM format). An identifying certificate is required for secure communication. The string must be sorted starting with the certificate to the highest level (root CA). Either certificate_chain *or* certificate_chain_file is required. You must set exactly one of these. Do not set both of them (this would produce a configuration error.
tls.identity.certificate_chain_file	A string that specifies the name of a file containing an identifying certificate chain (in PEM format). An identifying certificate is required for secure communication. The file must be sorted starting with the certificate to the highest level (root CA). Optionally, a private key may be appended to this file. If a private key is not appended to this file, then either private_key or private_key_file is required. Either certificate_chain *or* certificate_chain_file is required. You must set exactly ONE of these. Do not set both of them (this would produce a configuration error.
tls.identity.private_key_password	A string that specifies the password for private key.
tls.identity.private_key	A string containing a private key (in PEM format). Either private_key *or* private_key_file may be specified. Do not set both of them (this would produce a configuration error). If both are unspecified (NULL), the private key must be appended to the certificate chain file.
tls.identity.private_key_file	A string that specifies the name of a file containing a private key (in PEM format). Either private_key *or* private_key_file may be specified. Do not set both of them (this would produce a configuration error). If both are unspecified (NULL), the private key must be appended to the certificate chain file.

1. Assuming you used 'dds.transport.TCPv4.tcp1' as the alias to load the plugin. If not, change the prefix to match the string used with dds.transport.load_plugins.